# Table of contents

## 1 快速入门
1.1 概述 ........................................... 3  
1.2 准备工作  ........................ .......... 3  
1.2.1 硬件： ..................................... 3  
1.2.2 软件： .................................... 91  
1.3 安装 .......................................... 92  
1.3.1 IDE .......................................... 92  
1.3.2 手动安装 .................................. 92  
1.4 编译第一个工程 .............................. 122  

## 2 API 参考
2.1 API Conventions ................................. 123  
2.1.1 Error handling .............................. 123  
2.1.2 Configuration structures .................... 123  
2.1.3 Private APIs ................................ 124  
2.1.4 Components in example projects ............ 124  
2.1.5 API Stability ................................ 125  
2.2 应用层协议 ................................. 126  
2.2.1 ASIO port ................................ 126  
2.2.2 ESP-Modbus ................................ 126  
2.2.3 ESP-MQTT ................................ 127  
2.2.4 ESP-TLS ................................ 142  
2.2.5 ESP HTTP Client ......................... 157  
2.2.6 ESP Local Control ......................... 172  
2.2.7 ESP Serial Slave Link .................... 181  
2.2.8 ESP x509 Certificate Bundle .......... 198  
2.2.9 HTTP 服务器 ............................ 200  
2.2.10 HTTPS 服务器 ......................... 226  
2.2.11 ICMP Echo ................................ 230  
2.2.12 mDNS 服务 ............................ 234  
2.2.13 Mbed TLS ............................... 235  
2.2.14 IP 网络层协议 ......................... 236  
2.3 蓝牙 API ................................. 237  
2.3.1 BT COMMON ............................. 237  
2.3.2 BT LE ................................. 242  
2.3.3 CLASSIC BT ............................ 365  
2.3.4 Controller & VHCI ...................... 485  
2.3.5 ESP-BLE-MESH ......................... 494  
2.3.6 NimBLE-based host APIs .............. 776  
2.4 Error Codes Reference ................. 778  
2.5 连网 API ................................. 785  
2.5.1 Wi-Fi ................................. 785  
2.5.2 以太网 ................................. 888  
2.5.3 Thread ................................. 920  
2.5.4 IP 网络层协议 ......................... 926
2.5.5 外设 API .......................... 957
2.5.6 应用层协议 .......................... 959
2.6 外设 API .......................... 959
2.6.1 Analog to Digital Converter (ADC) Oneshot Mode Driver .......................... 959
2.6.2 Analog to Digital Converter (ADC) Continuous Mode Driver .......................... 968
2.6.3 Analog to Digital Converter (ADC) Calibration Driver .......................... 976
2.6.4 Clock Tree .......................... 980
2.6.5 Digital To Analog Converter (DAC) .......................... 988
2.6.6 GPIO & RTC GPIO .......................... 1001
2.6.7 通用定时器 .......................... 1020
2.6.8 I2C 驱动程序 .......................... 1034
2.6.9 Inter-IC Sound (I2S) .......................... 1051
2.6.10 LCD .......................... 1086
2.6.11 LED PWM 控制器 .......................... 1099
2.6.12 Motor Control Pulse Width Modulator (MCPWM) .......................... 1116
2.6.13 脉冲计数器 (PCNT) .......................... 1164
2.6.14 Remote Control Transceiver (RMT) .......................... 1178
2.6.15 SD Pull-up Requirements .......................... 1202
2.6.16 SDMMC Host Driver .......................... 1206
2.6.17 SD SPI Host Driver .......................... 1211
2.6.18 SDIO Card Slave Driver .......................... 1217
2.6.19 Sigma-Delta Modulation (SDM) .......................... 1227
2.6.20 SPI Master Driver .......................... 1232
2.6.21 SPI 从机驱动程序 .......................... 1257
2.6.22 ESP32-WROOM-32SE (Secure Element) .......................... 1264
2.6.23 驱动程序 .......................... 1274
2.6.24 Two-Wire Automotive Interface (TWAI) .......................... 1282
2.6.25 Universal Asynchronous Receiver/Transmitter (UART) .......................... 1300
2.7 Project Configuration .......................... 1325
2.7.1 Introduction .......................... 1325
2.7.2 Project Configuration Menu .......................... 1326
2.7.3 Using sdkconfig.defaults .......................... 1326
2.7.4 Kconfig Formatting Rules .......................... 1326
2.7.5 Backward Compatibility of Kconfig Options .......................... 1326
2.7.6 Configuration Options Reference .......................... 1326
2.8 配网 API .......................... 1600
2.8.1 Protocol Communication .......................... 1600
2.8.2 Unified Provisioning .......................... 1615
2.8.3 Wi-Fi Provisioning .......................... 1619
2.9 存储 API .......................... 1639
2.9.1 FAT 文件系统 .......................... 1639
2.9.2 量产程序 .......................... 1646
2.9.3 非易失性存储库 .......................... 1650
2.9.4 NVS 分区生成程序 .......................... 1672
2.9.5 NVS 分区解析程序 .......................... 1676
2.9.6 SD/SDIO/MMC 驱动程序 .......................... 1677
2.9.7 SPI Flash API .......................... 1690
2.9.8 SPIFFS 文件系统 .......................... 1726
2.9.9 虚拟文件系统组件 .......................... 1730
2.9.10 摩损均衡 API .......................... 1745
2.10 System API .......................... 1749
2.10.1 App Image Format .......................... 1749
2.10.2 Application Level Tracing .......................... 1755
2.10.3 Call function with external stack .......................... 1760
2.10.4 Chip Revision .......................... 1761
2.10.5 控制台终端 .......................... 1764
2.10.6 eFuse Manager .......................... 1773
2.10.7 Error Codes and Helper Functions .......................... 1793
### 3 H/W 硬件参考

| 2.10.8  | ESP HTTPS OTA                                      | 1796 |
| 2.10.9  | Event Loop Library                                 | 1802 |
| 2.10.10 | FreeRTOS                                           | 1814 |
| 2.10.11 | FreeRTOS Supplemental Features                    | 1927 |
| 2.10.12 | Heap Memory Allocation                             | 1946 |
| 2.10.13 | Heap Memory Debugging                              | 1960 |
| 2.10.14 | High Resolution Timer (ESP Timer)                  | 1971 |
| 2.10.15 | Internal and Unstable APIs                         | 1977 |
| 2.10.16 | Inter-Processor Call                               | 1979 |
| 2.10.17 | Interrupt allocation                               | 1984 |
| 2.10.18 | Logging library                                    | 1990 |
| 2.10.19 | 杂项系统 API                                       | 1997 |
| 2.10.20 | 空中升级 (OTA)                                     | 2012 |
| 2.10.21 | Performance Monitor                                | 2023 |
| 2.10.22 | 电源管理                                           | 2026 |
| 2.10.23 | POSIX Threads Support                              | 2032 |
| 2.10.24 | Random Number Generation                           | 2036 |
| 2.10.25 | 睡眠模式                                           | 2038 |
| 2.10.26 | SoC Capabilities                                   | 2050 |
| 2.10.27 | 系统时间                                           | 2059 |
| 2.10.28 | The himem allocation API                           | 2063 |
| 2.10.29 | ULP 协处理器编程                                  | 2066 |
| 2.10.30 | Watchdogs                                          | 2098 |

### 4 API 指南

| 4.1         | 应用层跟踪库                                      | 2107 |
| 4.1.1       | 概述                                              | 2107 |
| 4.1.2       | 运行模式                                           | 2107 |
| 4.1.3       | 配置选项与依赖项                                  | 2108 |
| 4.1.4       | 如何使用此库                                      | 2109 |
| 4.2         | 应用程序的启动流程                                | 2117 |
| 4.2.1       | 一级引导程序                                      | 2117 |
| 4.2.2       | 二级引导程序                                      | 2117 |
| 4.2.3       | 应用程序启动阶段                                  | 2118 |
| 4.3         | BluFi                                              | 2119 |
| 4.3.1       | 概览                                              | 2120 |
| 4.3.2       | BluFi 流程                                         | 2120 |
| 4.3.3       | BluFi 流程图                                       | 2120 |
| 4.3.4       | BluFi 中定义的帧格式                              | 2120 |
| 4.3.5       | ESP32 端的安全实现                                | 2125 |
| 4.3.6       | GATT 相关说明                                      | 2126 |
| 4.4         | 引导加载程序 (Bootloader)                          | 2126 |
| 4.4.1       | 引导加载程序兼容性                                | 2126 |
| 4.4.2       | 日志级别                                           | 2127 |
| 4.4.3       | 恢复出厂设置                                      | 2127 |
| 4.4.4       | 从测试固件启动                                    | 2128 |
| 4.4.5       | 回滚                                              | 2128 |
| 4.4.6       | 看门狗                                            | 2128 |
| 4.4.7       | 引导加载程序大小                                  | 2129 |
| 4.4.8       | 从深度睡眠中快速启动                             | 2129 |
| 4.4.9       | 自定义引导加载程序                                | 2129 |

### 4.5 构建系统

| 4.5.1       | 概述                                              | 2129 |
| 4.5.2       | 使用构建系统                                      | 2130 |
| 4.5.3       | 示例项目                                           | 2132 |
| 4.5.4       | 项目 CMakeLists 文件                              | 2132 |
4.5.5 组件 CMakeLists 文件 ........................................... 2134
4.5.6 组件配置 ....................................................... 2136
4.5.7 遵循器定义 ....................................................... 2136
4.5.8 组件依赖 ....................................................... 2136
4.5.9 覆盖项目部分设置 ................................................ 2140
4.5.10 仅配置组件 ..................................................... 2140
4.5.11 CMake 调试 ..................................................... 2141
4.5.12 组件 CMakeLists 示例 ............................................. 2141
4.5.13 自定义 sdkconfig 的默认值 ..................................... 2145
4.5.14 Flash 参数 ...................................................... 2145
4.5.15 构建 Bootloader ................................................ 2146
4.5.16 编写纯 CMake 组件 ............................................. 2146
4.5.17 组件中使用第三方 CMake 项目 .................................. 2146
4.5.18 组件中使用预建库 .............................................. 2147
4.5.19 在自定义 CMake 项目中使用 ESP-IDF ............................ 2147
4.5.20 ESP-IDF CMake 构建系统 API ................................... 2148
4.5.21 文件编译 & 增量构建 .......................................... 2152
4.5.22 构建系统的元数据 ............................................. 2152
4.5.23 构建系统内部 ................................................. 2153
4.5.24 从 ESP-IDF GNU Make 构建系统迁移到 CMake 构建系统 .... 2154
4.6 Core Dump ............................................................ 2155
4.6.1 Overview .......................................................... 2156
4.6.2 Configurations ..................................................... 2156
4.6.3 Save core dump to flash ........................................... 2157
4.6.4 Print core dump to UART .......................................... 2157
4.6.5 ROM Functions in Backtraces ..................................... 2157
4.6.6 Dumping variables on demand ..................................... 2158
4.6.7 Running idf.py coredump-info and idf.py coredump-debug ..... 2158
4.7 Deep Sleep Wake Stubs ................................................ 2160
4.7.1 Rules for Wake Stubs ................................................ 2160
4.7.2 Implementing A Stub ............................................... 2161
4.7.3 Loading Code Into RTC Memory .................................... 2161
4.7.4 Loading Data Into RTC Memory .................................... 2161
4.7.5 CRC Check For Wake Stubs ......................................... 2162
4.8 错误处理 ............................................................ 2162
4.8.1 概述 ............................................................. 2162
4.8.2 错误码 .......................................................... 2163
4.8.3 错误码到错误消息 ................................................ 2163
4.8.4 ESP_ERROR_CHECK 宏 ............................................. 2163
4.8.5 ESP_ERROR_CHECKWITHOUT_ABORT 宏 ..................... 2164
4.8.6 ESP_RETURN_ON_ERROR 宏 ....................................... 2164
4.8.7 ESP_GOTO_ON_ERROR 宏 ......................................... 2164
4.8.8 ESP_RETURN_ON_FALSE 宏 ....................................... 2164
4.8.9 ESP_GOTO_ON_FALSE 宏 ......................................... 2164
4.8.10 CHECK 宏使用示例 .............................................. 2164
4.8.11 错误处理模式 .................................................. 2165
4.8.12 C++ 异常 ...................................................... 2165
4.9 ESP-BLE-MESH ....................................................... 2166
4.9.1 ESP-BLE-MESH 快速入门 ......................................... 2166
4.9.2 ESP-BLE-MESH 示例 .............................................. 2171
4.9.3 ESP-BLE-MESH 演示视频 ......................................... 2173
4.9.4 ESP-BLE-MESH 常见问题手册 .................................... 2173
4.9.5 相关文档 ....................................................... 2173
4.10 ESP-IDF FreeRTOS (SMP) ........................................... 2200
4.10.1 Overview ......................................................... 2201
4.10.2 Symmetric Multiprocessing ...................................... 2201
4.10.3 Tasks ............................................................ 2202
4.10.4 SMP Scheduler .................................................... 2203
4.10.5 Critical Sections .................................................. 2207
4.10.6 Misc ................................................................. 2208
4.11 ESP-WIFI-MESH ....................................................... 2209
  4.11.1 概述 .............................................................. 2209
  4.11.2 简介 .............................................................. 2209
  4.11.3 ESP-WIFI-MESH 概念 ........................................... 2210
  4.11.4 建立网络 ......................................................... 2215
  4.11.5 管理网络 .......................................................... 2219
  4.11.6 数据传输 .......................................................... 2222
  4.11.7 信道切换 .......................................................... 2224
  4.11.8 性能 .............................................................. 2226
  4.11.9 更多注意事项 .................................................... 2227
4.12 Event Handling ....................................................... 2227
  4.12.1 Wi-Fi, Ethernet, and IP Events .................................. 2227
  4.12.2 Mesh Events ......................................................... 2228
  4.12.3 Bluetooth Events .................................................. 2229
4.13 严重错误 ............................................................. 2229
  4.13.1 概述 .............................................................. 2229
  4.13.2 紧急处理程序 ...................................................... 2230
  4.13.3 寄存器转储与回溯 ............................................... 2230
  4.13.4 GDB Stub ........................................................... 2232
  4.13.5 RTC 看门狗超时 .................................................. 2233
  4.13.6 Guru Meditation 错误 ............................................ 2233
  4.13.7 其他严重错误 .................................................... 2234
4.14 Flash 加密 .......................................................... 2237
  4.14.1 概述 .............................................................. 2237
  4.14.2 相关 eFuses ....................................................... 2237
  4.14.3 Flash 的加密过程 ............................................... 2238
  4.14.4 Flash 加密设置 ................................................... 2239
  4.14.5 可能出现的错误 .................................................. 2244
  4.14.6 ESP32 flash 加密状态 ............................................. 2246
  4.14.7 在加密的 flash 中读写数据 ..................................... 2246
  4.14.8 更新加密的 flash ............................................... 2247
  4.14.9 关闭 flash 加密 .................................................. 2247
  4.14.10 Flash 加密的要点 ............................................... 2247
  4.14.11 Flash 加密的局限性 ............................................ 2248
  4.14.12 Flash 加密与安全启动 .......................................... 2248
  4.14.13 Flash 加密的高级功能 ......................................... 2248
  4.14.14 技术细节 ......................................................... 2250
4.15 Hardware Abstraction ............................................... 2251
  4.15.1 Architecture ....................................................... 2251
  4.15.2 LL (Low Level) Layer ............................................. 2252
  4.15.3 HAL (Hardware Abstraction Layer) ............................. 2253
4.16 High-Level Interrupts ............................................... 2254
  4.16.1 Interrupt Levels ................................................... 2254
  4.16.2 Notes .............................................................. 2255
4.17 JTAG 调试 .......................................................... 2255
  4.17.1 引言 .............................................................. 2255
  4.17.2 工作原理 ......................................................... 2256
  4.17.3 选择 JTAG 适配器 ............................................... 2256
  4.17.4 安装 OpenOCD ..................................................... 2257
  4.17.5 配置 ESP32 目标板 ............................................... 2257
  4.17.6 启动调试器 ....................................................... 2262
  4.17.7 调试范例 ........................................................ 2263
  4.17.8 从源码构建 OpenOCD .......................................... 2263
  4.17.9 注意事项和补充内容 ............................................ 2267
  4.17.10 相关文档 ........................................................ 2272
4.18 链接器脚本生成机制 ............................................... 2297
4.18.1 概述 .................................................. 2297
4.18.2 快速上手 ............................................. 2297
4.18.3 链接器脚本生成机制介绍 ................................ 2300
4.19 lwIP .................................................. 2306
  4.19.1 Supported APIs ...................................... 2306
  4.19.2 BSD Sockets API ...................................... 2306
  4.19.3 Netconn API .......................................... 2310
  4.19.4 lwIP FreeRTOS Task .................................... 2310
  4.19.5 IPv6 Support ........................................... 2311
  4.19.6 esp-lwip custom modifications .......................... 2311
  4.19.7 Performance Optimization .............................. 2313
4.20 存储器类型 ........................................... 2314
  4.20.1 DRAM（数据 RAM） ..................................... 2314
  4.20.2 IRAM（指令 RAM） ..................................... 2315
  4.20.3 IROM（代码从 flash 中运行） ........................... 2315
  4.20.4 DROM（数据存储在 flash 中） .......................... 2316
  4.20.5 RTC Slow memory（RTC 慢速存储器） ................. 2316
  4.20.6 RTC FAST memory（RTC 快速存储器） ............... 2316
  4.20.7 具备 DMA 功能 ........................................ 2316
  4.20.8 在堆栈中放置 DMA 缓冲区 .............................. 2317
4.21 OpenThread ............................................ 2317
  4.21.1 Mode of the OpenThread stack .......................... 2317
  4.21.2 How To Write an OpenThread Application ................. 2318
  4.21.3 The OpenThread border router ......................... 2319
4.22 分区表 .............................................. 2319
  4.22.1 概述 .................................................. 2319
  4.22.2 内置分区表 .......................................... 2320
  4.22.3 创建自定义分区表 ..................................... 2320
  4.22.4 生成二进制分区表 ..................................... 2322
  4.22.5 分区大小检查 ......................................... 2323
  4.22.6 烧写分区表 .......................................... 2323
  4.22.7 分区工具 (parttool.py) ............................... 2323
4.23 Performance .......................................... 2325
  4.23.1 How to Optimize Performance ............................ 2325
  4.23.2 Guides ................................................. 2325
4.24 RF calibration ......................................... 2343
  4.24.1 Partial calibration ...................................... 2343
  4.24.2 Full calibration ........................................ 2343
  4.24.3 No calibration ......................................... 2343
  4.24.4 PHY initialization data .................................. 2343
  4.24.5 API Reference ........................................... 2344
4.25 Security .............................................. 2346
  4.25.1 Goals .................................................. 2347
  4.25.2 Platform Security ....................................... 2347
  4.25.3 Network Security ....................................... 2348
  4.25.4 Product Security ....................................... 2348
  4.25.5 Security Policy ........................................ 2349
4.26 Secure Boot ........................................... 2350
  4.26.1 Background ............................................. 2350
  4.26.2 Secure Boot Process Overview ............................ 2350
  4.26.3 Keys .................................................. 2351
  4.26.4 Bootloader Size ........................................ 2351
  4.26.5 How To Enable Secure Boot .............................. 2351
  4.26.6 Re-Flashable Software Bootloader ....................... 2352
  4.26.7 Generating Secure Boot Signing Key ..................... 2353
  4.26.8 Remote Signing of Images ............................... 2353
  4.26.9 Secure Boot Best Practices ............................. 2354
  4.26.10 Technical Details ...................................... 2354
4.26.11 Secure Boot & Flash Encryption ........................................... 2355
4.26.12 Signed App Verification Without Hardware Secure Boot ............ 2355
4.26.13 Advanced Features ......................................................... 2356
4.27 Secure Boot V2 ................................................................. 2356
  4.27.1 Background ............................................................... 2356
  4.27.2 Advantages ............................................................... 2357
  4.27.3 Secure Boot V2 Process ............................................... 2357
  4.27.4 Signature Block Format ............................................... 2357
  4.27.5 Verifying a Signature Block ........................................... 2358
  4.27.6 Verifying an Image ..................................................... 2358
  4.27.7 Bootloader Size ........................................................ 2358
  4.27.8 eFuse usage .............................................................. 2358
  4.27.9 How To Enable Secure Boot V2 ....................................... 2359
  4.27.10 Restrictions after Secure Boot is enabled ............................ 2360
  4.27.11 Generating Secure Boot Signing Key ................................ 2360
  4.27.12 Remote Signing of Images ............................................ 2360
  4.27.13 Secure Boot Best Practices ......................................... 2360
  4.27.14 Technical Details ...................................................... 2361
  4.27.15 Secure Boot & Flash Encryption .................................... 2361
  4.27.16 Signed App Verification Without Hardware Secure Boot ........... 2361
  4.27.17 Advanced Features ..................................................... 2362
4.28 片外 RAM ................................................................. 2362
  4.28.1 简介 ................................................................. 2362
  4.28.2 硬件 ................................................................. 2362
  4.28.3 配置片外 RAM ....................................................... 2362
  4.28.4 片外 RAM 使用限制 ................................................ 2364
  4.28.5 初始化失败 ........................................................... 2364
  4.28.6 Chip revisions .......................................................... 2364
4.29 Thread Local Storage ....................................................... 2365
  4.29.1 Overview ............................................................... 2365
  4.29.2 FreeRTOS Native API .................................................. 2365
  4.29.3 Pthread API ............................................................. 2365
  4.29.4 C11 Standard ........................................................... 2366
4.30 工具 ................................................................. 2366
  4.30.1 IDF Frontend - idf.py ................................................ 2366
  4.30.2 IDF Docker Image ..................................................... 2370
  4.30.3 IDF Windows Installer ............................................... 2371
  4.30.4 IDF Component Manager .............................................. 2372
  4.30.5 IDF Clang Tidy ........................................................ 2374
  4.30.6 Downloadable Tools .................................................... 2375
4.31 ESP32 中的单元测试 .................................................. 2388
  4.31.1 添加常规测试用例 ................................................... 2388
  4.31.2 添加多设备测试用例 ................................................ 2389
  4.31.3 添加多阶段测试用例 ................................................ 2389
  4.31.4 应用于不同芯片的单元测试 ....................................... 2390
  4.31.5 编译单元测试程序 ................................................ 2391
  4.31.6 运行单元测试 ........................................................ 2391
  4.31.7 带缓存补充定时器的时代表码 ..................................... 2392
  4.31.8 Mocks ................................................................. 2393
4.32 Unit Testing on Linux ..................................................... 2394
  4.32.1 Embedded Software Tests .............................................. 2395
  4.32.2 IDF Unit Tests on Linux Host ....................................... 2395
4.33 Wi-Fi 驱动程序 ......................................................... 2396
  4.33.1 ESP32 Wi-Fi 功能列表 ............................................... 2396
  4.33.2 如何编写 Wi-Fi 应用程序 .......................................... 2396
  4.33.3 ESP32 Wi-Fi API 错误代码 .......................................... 2397
  4.33.4 初始化 ESP32 Wi-Fi API 参数 ..................................... 2397
  4.33.5 ESP32 Wi-Fi 编程模型 ............................................... 2397

vii
5.1.2 esp-idf-cxx Component

4.33.6 ESP32 Wi-Fi 事件描述
4.33.7 ESP32 Wi-Fi station 一般情况
4.33.8 ESP32 Wi-Fi AP 一般情况
4.33.9 ESP32 Wi-Fi 扫描
4.33.10 ESP32 Wi-Fi station 连接场景
4.33.11 找到多个 AP 时的 ESP32 Wi-Fi station 连接
4.33.12 Wi-Fi 重新连接
4.33.13 Wi-Fi beacon 超时
4.33.14 ESP32 Wi-Fi 配置
4.33.15 Wi-Fi Easy Connect™ (DPP)
4.33.16 无线网络管理
4.33.17 无线资源管理
4.33.18 ESP32 Wi-Fi 节能模式
4.33.19 ESP32 Wi-Fi 吞吐量
4.33.20 Wi-Fi 80211 数据包发送
4.33.21 Wi-Fi Sniffer 模式
4.33.22 Wi-Fi 多根天线
4.33.23 Wi-Fi 信道状态信息
4.33.24 Wi-Fi 信道状态信息配置
4.33.25 Wi-Fi HT20/40
4.33.26 Wi-Fi QoS
4.33.27 Wi-Fi AMSDU
4.33.28 Wi-Fi 分片
4.33.29 WPS 注册
4.33.30 Wi-Fi 缓冲区使用情况
4.33.31 如何提高 Wi-Fi 性能
4.33.32 Wi-Fi Menuconfig
4.33.33 故障排除

4.34 Wi-Fi Security
4.34.1 ESP32 Wi-Fi Security Features
4.34.2 Protected Management Frames (PMF)
4.34.3 WPA3-Personal

4.35 RF 共存
4.35.1 概览
4.35.2 ESP32 支持的共存场景
4.35.3 共存机制与策略
4.35.4 如何使用共存功能

4.36 Reproducible Builds
4.36.1 Introduction
4.36.2 Reasons for non-reproducible builds
4.36.3 Enabling reproducible builds in ESP-IDF
4.36.4 How reproducible builds are achieved
4.36.5 Reproducible builds and debugging
4.36.6 Factors which still affect reproducible builds

4.37 C++ Support
4.37.1 esp-idf-cxx Component
4.37.2 C++ language standard
4.37.3 Multithreading
4.37.4 Exception handling
4.37.5 Runtime Type Information (RTTI)
4.37.6 Developing in C++
4.37.7 Limitations
4.37.8 What to Avoid

5 迁移指南

5.1 迁移到 ESP-IDF 5.x
5.1.1 从 4.4 迁移到 5.0
5.1.2 从 5.0 迁移到 5.1
# Libraries and Frameworks

## 6.1 Cloud Frameworks

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.1</td>
<td>ESP RainMaker</td>
</tr>
<tr>
<td>6.1.2</td>
<td>AWS IoT</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Azure IoT</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Google IoT Core</td>
</tr>
<tr>
<td>6.1.5</td>
<td>Aliyun IoT</td>
</tr>
<tr>
<td>6.1.6</td>
<td>Joylink IoT</td>
</tr>
<tr>
<td>6.1.7</td>
<td>Tencent IoT</td>
</tr>
<tr>
<td>6.1.8</td>
<td>Tencentyun IoT</td>
</tr>
<tr>
<td>6.1.9</td>
<td>Baidu IoT</td>
</tr>
</tbody>
</table>

## 6.2 Espressif's Frameworks

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.1</td>
<td>Espressif Audio Development Framework</td>
</tr>
<tr>
<td>6.2.2</td>
<td>ESP-CSI</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Espressif DSP Library</td>
</tr>
<tr>
<td>6.2.4</td>
<td>ESP-WIFI-MESH Development Framework</td>
</tr>
<tr>
<td>6.2.5</td>
<td>ESP-WHO</td>
</tr>
<tr>
<td>6.2.6</td>
<td>ESP RainMaker</td>
</tr>
<tr>
<td>6.2.7</td>
<td>ESP-IoT-Solution</td>
</tr>
<tr>
<td>6.2.8</td>
<td>ESP-Protocols</td>
</tr>
<tr>
<td>6.2.9</td>
<td>ESP-BSP</td>
</tr>
<tr>
<td>6.2.10</td>
<td>ESP-IDF-CXX</td>
</tr>
</tbody>
</table>

# Contributions Guide

## 7.1 How to Contribute

## 7.2 Before Contributing

## 7.3 Pull Request Process

## 7.4 Legal Part

## 7.5 Related Documents

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5.1</td>
<td>Espressif IoT Development Framework Style Guide</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Install pre-commit Hook for ESP-IDF Project</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Documenting Code</td>
</tr>
<tr>
<td>7.5.4</td>
<td>创建示例项目</td>
</tr>
<tr>
<td>7.5.5</td>
<td>API Documentation Template</td>
</tr>
<tr>
<td>7.5.6</td>
<td>Contributor Agreement</td>
</tr>
<tr>
<td>7.5.7</td>
<td>Copyright Header Guide</td>
</tr>
<tr>
<td>7.5.8</td>
<td>ESP-IDF Tests with Pytest Guide</td>
</tr>
</tbody>
</table>

# ESP-IDF 版本简介

## 8.1 发布版本

## 8.2 我该选择哪个版本？

## 8.3 版本管理

## 8.4 支持期限

## 8.5 查看当前版本

## 8.6 Git 工作流

## 8.7 更新 ESP-IDF

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.7.1</td>
<td>更新至一个稳定发布版本</td>
</tr>
<tr>
<td>8.7.2</td>
<td>更新至一个预发布版本</td>
</tr>
<tr>
<td>8.7.3</td>
<td>更新至 master 分支</td>
</tr>
<tr>
<td>8.7.4</td>
<td>更新至一个发布分支</td>
</tr>
</tbody>
</table>

# 资源

## 9.1 PlatformIO

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.1</td>
<td>What is PlatformIO?</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Installation</td>
</tr>
<tr>
<td>9.1.3</td>
<td>Configuration</td>
</tr>
<tr>
<td>9.1.4</td>
<td>Tutorials</td>
</tr>
<tr>
<td>9.1.5</td>
<td>Project Examples</td>
</tr>
<tr>
<td>9.1.6</td>
<td>Next Steps</td>
</tr>
</tbody>
</table>

---

ix
10 Copyrights and Licenses

10.1 Software Copyrights
10.1.1 Firmware Components
10.1.2 Documentation

10.2 ROM Source Code Copyrights
10.3 Xtensa libhal MIT License
10.4 TinyBasic Plus MIT License
10.5 TJpgDec License
这里是乐鑫 IoT 开发框架 (esp-idf) 的文档中心。ESP-IDF 是 ESP32、ESP32-S 和 ESP32-C 系列芯片的官方开发框架。

本文档仅包含针对 ESP32 芯片的 ESP-IDF 使用。

| 快速入门 | API 参考 | API 指南 |
Chapter 1

快速入门

本文档旨在指导用户搭建 ESP32 硬件开发的软件环境，通过一个简单的示例展示如何使用 ESP-IDF (Espressif IoT Development Framework) 配置菜单，并编译、下载固件至 ESP32 开发板等步骤。

备注：这是 ESP-IDF master 分支（最新版本）的文档，该版本在持续开发中。还有 Stable version 的文档，以及其他版本的文档 ESP-IDF 版本简介 供参考。

1.1 概述

ESP32 SoC 芯片支持以下功能:

- 2.4 GHz Wi-Fi
- 蓝牙
- 高性能 Xtensa® 32 位 LX6 双核处理器
- 超低功耗处理器
- 多种外设

ESP32 采用 40 nm 工艺制成，具有最佳的功耗性能、射频性能、稳定性、通用性和可靠性，适用于各种应用场景和不同功耗需求。

乐鑫为用户提供完整的软、硬件资源，进行 ESP32 硬件设备的开发。其中，乐鑫的软件开发环境 ESP-IDF 旨在协助用户快速开发物联网 (IoT) 应用，可满足用户对 Wi-Fi、蓝牙、低功耗等方面的要求。

1.2 准备工作

1.2.1 硬件:

- 一款 ESP32 开发板
- USB 数据线 (A 转 Micro-B)
- 电脑 (Windows、Linux 或 macOS)

备注：目前一些开发板使用的是 USB Type C 接口。请确保使用合适的数据线来连接开发板！

以下是 ESP32 官方开发板，点击链接可了解更多硬件信息。
ESP32-DevKitC V4 入门指南

本指南介绍了如何开始使用 ESP32-DevKitC V4 开发板。

准备工作

- ESP32-DevKitC V4 开发板
- USB A / micro USB B 数据线
- PC（Windows、Linux 或 Mac OS）

您可以跳过介绍部分，直接前往应用程序开发 章节。

概述  ESP32-DevKitC V4 是 乐鑫 一款基于 ESP32 的小型开发板，板上模组的绝大部分管脚均已引出，开发人员可根据实际需求，轻松通过跳线连接多种外围器件，或将开发板插在面包板上使用。

为了更好地满足不同用户需求，ESP32-DevKitC V4 还支持以下不同配置：

- 可选多款 ESP32 模组
  - ESP32-WROOM-32E
  - ESP32-WROOM-32UE
  - ESP32-WROOM-32D
  - ESP32-WROOM-32U
  - ESP32-SOLO-1
  - ESP32-WROVER-E
  - ESP32-WROVER-IE
  - ESP32-WROOM-DA

- 可选排针或排母

详情请见 乐鑫产品选型工具。

功能说明  ESP32-DevKitC V4 开发板的主要组件、接口及控制方式见下。

图 1: ESP32-DevKitC V4（板载 ESP32-WROOM-32）
**主要组件**

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-WROOM-32</td>
<td>基于 ESP32 的模组。更多详情，请见《ESP32-WROOM-32 技术规格书》。</td>
</tr>
<tr>
<td>Boot</td>
<td>复位按钮。</td>
</tr>
</tbody>
</table>
| USB-to-UART 桥接器 | 下载按键，按住 Boot 键并保持。同时按一下 EN 键（此时不要松开 Boot 键）进入“固件下载”模式，通过串口下载固件。
| Micro USB 接口  | USB 接口，可用作电路板的供电电源，或连接 PC 和 ESP32-WROOM-32 模组的通信接口。 |
| 5V Power On LED | 开发板通电后（USB 或外部 5 V），该指示灯将亮起。更多信息，请见相关文档中的原理图。 |
| I/O            | 板上模组的绝大部分管脚均已引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。 |

**电源选项** 开发板可从以下三种供电方式中任选其一：
- Micro USB 供电（默认）
- 5V / GND 管脚供电
- 3V3 / GND 管脚供电

**警告**：上述供电模式 不可同时连接，否则可能会损坏开发板和/或电源。

**排针** 下表列出了开发板两侧排针（J1 和 J3）的名称和功能，排针名称如图 ESP32-DevKitC V4 （或 ESP32-WROOM-32）中所示。

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3V3</td>
<td>P</td>
<td>3.3 V 电源</td>
</tr>
<tr>
<td>2</td>
<td>EN</td>
<td>I</td>
<td>CHIP PU, Reset</td>
</tr>
<tr>
<td>3</td>
<td>VP</td>
<td>I</td>
<td>GPIO36, ADC1_CH0, S_VP</td>
</tr>
<tr>
<td>4</td>
<td>VN</td>
<td>I</td>
<td>GPIO39, ADC1_CH3, S_VN</td>
</tr>
<tr>
<td>5</td>
<td>IO34</td>
<td>I</td>
<td>GPIO34, ADC1_CH6, VDET_1</td>
</tr>
<tr>
<td>6</td>
<td>IO35</td>
<td>I</td>
<td>GPIO35, ADC1_CH7, VDET_2</td>
</tr>
<tr>
<td>7</td>
<td>IO32</td>
<td>I/O</td>
<td>GPIO32, ADC1_CH4, TOUCH_CH9, XTAL_32K_P</td>
</tr>
<tr>
<td>8</td>
<td>IO33</td>
<td>I/O</td>
<td>GPIO33, ADC1_CH5, TOUCH_CH8, XTAL_32K_N</td>
</tr>
<tr>
<td>9</td>
<td>IO25</td>
<td>I/O</td>
<td>GPIO25, ADC1_CH8, DAC_1</td>
</tr>
<tr>
<td>10</td>
<td>IO26</td>
<td>I/O</td>
<td>GPIO26, ADC2_CH9, DAC_2</td>
</tr>
<tr>
<td>11</td>
<td>IO27</td>
<td>I/O</td>
<td>GPIO27, ADC2_CH7, TOUCH_CH7</td>
</tr>
<tr>
<td>12</td>
<td>IO14</td>
<td>I/O</td>
<td>GPIO14, ADC2_CH6, TOUCH_CH6, MTMS</td>
</tr>
<tr>
<td>13</td>
<td>IO12</td>
<td>I/O</td>
<td>GPIO12, ADC2_CH5, TOUCH_CH5, MTDI</td>
</tr>
<tr>
<td>14</td>
<td>GND</td>
<td>G</td>
<td>接地</td>
</tr>
<tr>
<td>15</td>
<td>IO13</td>
<td>I/O</td>
<td>GPIO13, ADC2_CH4, TOUCH_CH4, MTCK</td>
</tr>
<tr>
<td>16</td>
<td>D2</td>
<td>I/O</td>
<td>GPIO9, D2</td>
</tr>
<tr>
<td>17</td>
<td>D3</td>
<td>I/O</td>
<td>GPIO10, D3</td>
</tr>
<tr>
<td>18</td>
<td>CMD</td>
<td>I/O</td>
<td>GPIO11, CMD</td>
</tr>
<tr>
<td>19</td>
<td>5V</td>
<td>P</td>
<td>5 V 电源</td>
</tr>
</tbody>
</table>

1: P: 电源；I: 输入；O: 输出。
2: 管脚 D0、D1、D2、D3、CMD 和 CLK 用于 ESP32 芯片与 SPI flash 间的内部通信，集中分布在开发板两侧靠近 USB 端口的位置。通常而言，这些管脚最好不连，否则可能影响 SPI flash / SPI RAM 的工作。
### 管脚布局

**有关 C15 的提示**

较早版本 ESP32-DevKitC 开发板上的 C15 可能存在以下问题:

1. 管脚 GPIO16 和 GPIO17 仅适用于搭载 ESP32-WROOM 系列和 ESP32-SOLO-1 的开发板，搭载 ESP32-WROVER 系列开发板的管脚 GPIO16 和 GPIO17 保留内部使用。

#### 表 2: ESP32-DevKitC 管脚布局

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>G</td>
<td>接地</td>
</tr>
<tr>
<td>2</td>
<td>IO23</td>
<td>I/O</td>
<td>GPIO23</td>
</tr>
<tr>
<td>3</td>
<td>IO22</td>
<td>I/O</td>
<td>GPIO22</td>
</tr>
<tr>
<td>4</td>
<td>TX</td>
<td>I/O</td>
<td>GPIO1, U0TXD</td>
</tr>
<tr>
<td>5</td>
<td>RX</td>
<td>I/O</td>
<td>GPIO3, U0RXD</td>
</tr>
<tr>
<td>6</td>
<td>IO21</td>
<td>I/O</td>
<td>GPIO21</td>
</tr>
<tr>
<td>7</td>
<td>GND</td>
<td>G</td>
<td>接地</td>
</tr>
<tr>
<td>8</td>
<td>IO19</td>
<td>I/O</td>
<td>GPIO19</td>
</tr>
<tr>
<td>9</td>
<td>IO18</td>
<td>I/O</td>
<td>GPIO18</td>
</tr>
<tr>
<td>10</td>
<td>IO5</td>
<td>I/O</td>
<td>GPIO5</td>
</tr>
<tr>
<td>11</td>
<td>IO17</td>
<td>I/O</td>
<td>GPIO17</td>
</tr>
<tr>
<td>12</td>
<td>IO16</td>
<td>I/O</td>
<td>GPIO16</td>
</tr>
<tr>
<td>13</td>
<td>IO4</td>
<td>I/O</td>
<td>GPIO4, ADC2_CH0, TOUCH_CH0</td>
</tr>
<tr>
<td>14</td>
<td>IO0</td>
<td>I/O</td>
<td>GPIO0, ADC2_CH1, TOUCH_CH1, Boot</td>
</tr>
<tr>
<td>16</td>
<td>IO2</td>
<td>I/O</td>
<td>GPIO2, ADC2_CH2, TOUCH_CH2</td>
</tr>
<tr>
<td>17</td>
<td>IO15</td>
<td>I/O</td>
<td>GPIO15, ADC2_CH3, TOUCH_CH3, MTDO</td>
</tr>
<tr>
<td>17</td>
<td>D1</td>
<td>I/O</td>
<td>GPIO8, D1</td>
</tr>
<tr>
<td>18</td>
<td>D0</td>
<td>I/O</td>
<td>GPIO7, D0</td>
</tr>
<tr>
<td>19</td>
<td>CLK</td>
<td>I/O</td>
<td>GPIO6, CLK</td>
</tr>
</tbody>
</table>

#### 图 2: ESP32-DevKitC 管脚布局（点击放大）
- 开发板上电后可能进入下载模式；
- 如果用户通过 GPIO0 输出时钟，C15 可能会影响信号。

用户如果认为 C15 可能影响开发板的使用，则可以将 C15 完全移除。C15 在开发板上的具体位置见下图黄色部分。

图 3: C15（黄色）在 ESP32-DevKitC V4 开发板上的位置

应用程序开发  ESP32-DevKitC V4 上电前，请首先确认开发板完好无损。

现在，请前往 快速入门 中的 安装 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

图 4: ESP32-DevKitC 开发板尺寸（板载 ESP32-WROOM-32 模组）仰视图

开发板尺寸

相关文档
- ESP32-DevKitC V4 原理图 (PDF)
Chapter 1. 快速入门

- 《ESP32 技术规格书》(PDF)
- 《ESP32-WROOM-32 技术规格书》(PDF)
- 《ESP32-WROOM-32D & ESP32-WROOM-32U 技术规格书》(PDF)
- 《ESP32-WROOM-DA 技术规格书》(PDF)
- 《ESP32-WROVER 技术规格书》(PDF)
- 《ESP32-WROVER-B 技术规格书》(PDF)

有关本开发板的更多设计文档，请联系我们的商务部门 sales@espressif.com。

### ESP32-DevKitC V2 入门指南

本指南介绍了如何开始使用 ESP32-DevKitC V2 开发板。

#### 准备工作

- ESP32-DevKitC V2 开发板
- USB A / micro USB B 数据线
- PC（Windows，Linux 或 macOS）

您可以跳过介绍部分，直接前往应用程序开发 章节。

#### 概述

ESP32-DevKitC V2 是 乐鑫 一款基于 ESP32 的小型开发板，板上板的绝大部分管脚均已引出，开发人员可根据实际需求，轻松通过跳线连接多种外围器件，或将开发板插入面包板上使用。

#### 功能说明

ESP32-DevKitC V2 开发板的主要组件、接口及控制方式见下。

![ESP-WROOM-32](image)

图 5: ESP32-DevKitC V2 开发板
### 主要组件

<table>
<thead>
<tr>
<th>组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-WROOM-32</td>
<td>基于 ESP32 的模组，更多详情，请见《ESP32-WROOM-32 技术规格书》。</td>
</tr>
<tr>
<td>EN</td>
<td></td>
</tr>
<tr>
<td>Boot</td>
<td>下载按键。按下 Boot 键并保持，同时按下 EN 键（此时不要松开 Boot 键）进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>Micro USB 端口</td>
<td>USB 接口。可用作电路板的供电电源，或连接 PC 和 ESP32-WROOM-32 模组的通信接口。</td>
</tr>
<tr>
<td>I/O</td>
<td>板上模组的绝大部分管脚均已引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。</td>
</tr>
</tbody>
</table>

### 电源选项

开发板可从以下三种供电方式中任选其一：

- Micro USB 供电（默认）
- 5V / GND 管脚供电
- 3V3 / GND 管脚供电

警告：上述供电模式不可同时连接，否则可能会损坏开发板和/或电源。

### 应用程序开发

ESP32-DevKitC V2 上电前，请首先确认开发板完好无损。

现在，请前往快速入门 中的 安装 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

### 相关文档

- ESP32-DevKitC 原理图 (PDF)
- 《ESP32 技术规格书》 (PDF)
- 《ESP32-WROOM-32 技术规格书》 (PDF)

### ESP-WROVER-KIT V4.1 入门指南

本指南介绍了如何开始使用 ESP-WROVER-KIT V4.1 开发板及其功能和相关配置。

### 准备工作

- ESP-WROVER-KIT V4.1 开发板
- USB 2.0 数据线（A 转 Micro-B）
- PC（Windows、Linux 或 macOS）

您可以跳过介绍部分，直接前往应用程序开发 章节。

### 概述

ESP-WROVER-KIT 是乐鑫 一款基于 ESP32 的开发板。

ESP-WROVER-KIT 开发板已集成了如下组件：

- ESP32-WROVER-E 模组
- LCD 屏
- microSD 卡槽

此外，ESP-WROVER-KIT 的独特之处在于集成了两款先进多协议 USB 桥接器（FTDI FT2232HL），允许开发人员直接通过 USB 接口，使用 JTAG 对 ESP32 进行调试，无需额外的 JTAG 调试器。ESP-WROVER-KIT 可为开发人员提供简单、便捷且极具成本效益的开发体验。

为了便于使用，板上模组的绝大部分管脚均已引出至开发板的引脚。
备注：ESP32 的 GPIO16 和 GPIO17 管脚用作 PSRAM 的片选和时钟信号。默认情况下，为了给用户提供可靠的性能，这两个 GPIO 管脚不引出至开发板引脚。

功能概述 ESP-WROVER-KIT 开发板的主要组件和连接方式如下图所示。

图 6: ESP-WROVER-KIT 功能框图

功能说明 ESP-WROVER-KIT 开发板的主要组件、接口及控制方式见下。

图 7: ESP-WROVER-KIT 开发板布局--俯视图

下表介绍了开发板的主要组件，顺序如下：
- 从图片右上角开始，以顺时针顺序介绍了图 1 中的主要组件
- 然后以同样的顺序介绍了图 2 中的主要组件
图 8: ESP-WROVER-KIT 开发板布局—仰视图

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT2232HL</td>
<td>FT2232HL 为多协议 USB 转串口桥接器。开发人员可通过 USB 接口对 FT2232HL 芯片进行控制和编程，与 ESP32 建立连接。FT2232HL 芯片可在通道 A 提供 USB-to-JTAG 接口功能，并在通道 B 提供 USB-to-Serial 接口功能，便于开发人员的应用开发与调试。详见 ESP-WROVER-KIT V4.1 原理图。</td>
</tr>
<tr>
<td>32.768 kHz</td>
<td>32.768 kHz 振荡，可提供 Deep-sleep 下使用的低功耗时钟。</td>
</tr>
<tr>
<td>0 欧电阻</td>
<td>ESP-WROVER-KIT 开发板设计了一个 0 欧电阻，可从测量 ESP32 系列微模组在不同功耗模式下的电流，直接移除或替换为分流器。</td>
</tr>
<tr>
<td>ESP32-WROVER-E 模组</td>
<td>这款 ESP32 模组内置 64-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。</td>
</tr>
<tr>
<td>诊断 LED 信号灯</td>
<td>本开发板 FT2232HL 芯片的 GPIO 管脚连接了 4 个红色 LED 信号灯，以备后用。</td>
</tr>
<tr>
<td>UART</td>
<td>串口。FT2232HL 和 ESP32 的串行 TX/RX 信号已引出至 JP2 的两端。默认情况下，这两路信号由跳线帽连接。如果仅使用 ESP32 模组串口，则可移除相关跳线帽，将模组连接至其他外部串口设备。</td>
</tr>
<tr>
<td>SPI</td>
<td>默认情况下，ESP32 使用 SPI 接口内嵌 flash 和 PSRAM。使用这些引脚连接 ESP32 和其他 SPI 设备。这种情况下，需增加额外的片选 (CS) 信号。注意，本接口的工作电压为 3.3 V。</td>
</tr>
<tr>
<td>CTS/RTS</td>
<td>串口流控信号，管脚默认不连接至电路。为便能该功能，必需使用跳线帽断开 J14 的相应管脚。</td>
</tr>
<tr>
<td>JTAG</td>
<td>JTAG 接口，FT2232HL 和 ESP32 的 JTAG 信号已引出至 JP2 的两端。默认情况下，这两路信号不连接。如需使能 JTAG，请按照设置选项的介绍，连接跳线帽。</td>
</tr>
<tr>
<td>USB 端口</td>
<td>USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。</td>
</tr>
<tr>
<td>EN</td>
<td>复位按键。</td>
</tr>
<tr>
<td>BOOT 按键</td>
<td>下载按键，按下 Boot 键并保持，同时按一下 EN 键（此时不要松开 Boot 键）进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>电源开关</td>
<td>电源开关。拨向 Boot 按键一侧，开发板上电；拨离 Boot 按键一侧，开发板掉电。</td>
</tr>
<tr>
<td>电源选择开关</td>
<td>ESP-WROVER-KIT 开发板可通过 USB 端口或 5V 输入接口供电。用户可使用跳线帽在两种供电模式中进行选择。更多详细信息，请见章节设置选项中有关 JP7 连接器的描述。</td>
</tr>
<tr>
<td>5V Input</td>
<td>5V 电源接口。为标准同轴电源接口，5.5 x 2.1 mm，中心正极。建议仅在开发板自动运行（未连接 USB）时使用。</td>
</tr>
<tr>
<td>5V Power On LED</td>
<td>当开发板通电后，USB 转接器 5V 供电。该红色指示灯将亮起。</td>
</tr>
<tr>
<td>LDO</td>
<td>5V-to-3.3 低噪声线性稳压器 NCP1117(1A)。NCP1117 最大电流输出为 1 A。板上 LDO 为固定输出电压，但用户也可以选用具有可变输出电压的 LDO。更多信息，请见 ESP-WROVER-KIT V4.1 原理图。</td>
</tr>
</tbody>
</table>
设置选项 用户可通过 3 组排针，设置开发板功能，其中常见功能见下表：
<table>
<thead>
<tr>
<th>排针</th>
<th>读线设置</th>
<th>功能描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP7</td>
<td></td>
<td>使用外部电源为 ESP-WROVER-KIT 开发板供电</td>
</tr>
<tr>
<td></td>
<td></td>
<td>使用 USB 端口为 ESP-WROVER-KIT 开发板供电</td>
</tr>
<tr>
<td>JP2</td>
<td></td>
<td>使能 JTAG 功能</td>
</tr>
<tr>
<td></td>
<td></td>
<td>使能 UART 通信</td>
</tr>
</tbody>
</table>
Chapter 1. 快速入门

ESP32 管脚分配 ESP32 模组的部分管脚或终端已被板上组件占用或用于外部硬件设备。如果某管脚对应的特定硬件未连接，则该管脚可用作他用。比如，摄像头/JP4 排针未连接相应硬件，则这些 GPIO 可用于其他用途。

部分管脚具备多个功能，可供板上组件或外部硬件设备同时使用，比如 GPIO0 和 GPIO2。由于管脚限制，一些外围设备不可同时使用，比如，由于 JTAG 和 SD 卡槽需共用部分管脚，因此一些使用 SD 卡功能的应用无法同时进行 JTAG 调试。

其他情况下，不同外设可同时使用。比如，LCD 屏幕和 SD 卡仅共用一个 GPIO21 管脚，可以同时使用。该管脚可为 LCD 屏幕提供 D/C（数据/控制）信号，并用于读取来自 SD 卡槽的卡检测信号。如无需使用卡检测功能，开发人员还可以通过移除 R167 来禁用该功能。此时，LCD 和 SD 卡槽可同时使用。

更多外设共享管脚的介绍，请见下一章节中的表格。

主要 I/O 连接器 / JP1 JP1 连接器包括 14 x 2 个排针，具体功能可见下表中间 “I/O” 列的介绍。两侧的“共用” 列则介绍了这些管脚在板上的其他用途。

<table>
<thead>
<tr>
<th>共用</th>
<th>I/O</th>
<th>I/O</th>
<th>共用</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>3.3V</td>
<td>GND</td>
<td>n/a</td>
</tr>
<tr>
<td>NC/XTAL</td>
<td>IO32</td>
<td>IO33</td>
<td>NC/XTAL</td>
</tr>
<tr>
<td>JTAG，microSD</td>
<td>IO12</td>
<td>IO13</td>
<td>JTAG，microSD</td>
</tr>
<tr>
<td>JTAG，microSD</td>
<td>IO14</td>
<td>IO27</td>
<td>摄像头</td>
</tr>
<tr>
<td>摄像头</td>
<td>IO26</td>
<td>IO25</td>
<td>摄像头，LCD</td>
</tr>
<tr>
<td>摄像头</td>
<td>IO35</td>
<td>IO34</td>
<td>摄像头</td>
</tr>
<tr>
<td>JTAG</td>
<td>EN</td>
<td>IO23</td>
<td>摄像头，LCD</td>
</tr>
<tr>
<td>摄像头，LCD</td>
<td>IO22</td>
<td>IO21</td>
<td>摄像头，LCD，microSD</td>
</tr>
<tr>
<td>摄像头，LCD</td>
<td>IO19</td>
<td>IO18</td>
<td>摄像头，LCD</td>
</tr>
<tr>
<td>摄像头，LCD</td>
<td>IO5</td>
<td>IO17</td>
<td>PSRAM</td>
</tr>
<tr>
<td>PSRAM</td>
<td>IO16</td>
<td>IO4</td>
<td>LED，摄像头，microSD</td>
</tr>
<tr>
<td>摄像头，LED，Boot</td>
<td>IO0</td>
<td>IO2</td>
<td>LED，microSD</td>
</tr>
<tr>
<td>JTAG，microSD</td>
<td>IO15</td>
<td>5V</td>
<td></td>
</tr>
</tbody>
</table>

说明:
- NC/XTAL - 32.768 kHz 晶振
- JTAG - JTAG / JP2
- Boot - Boot 按键 / SW2
- 摄像头 - 摄像头 / JP4
- LED - RGB LED
- microSD - microSD Card / J4
- LCD - LCD / U5
- PSRAM - ESP32-WROVER-E 的 PSRAM

32.768 kHz 晶振

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO32</td>
</tr>
<tr>
<td>2</td>
<td>GPIO33</td>
</tr>
</tbody>
</table>

备注：默认情况下，管脚 GPIO32 和 GPIO33 已连接至晶振。因此，为了保证信号的完整性，这两个管脚并未连接至 JP1 I/O 连接器。用户可通过将 R11 或 R23 处的 0 欧电阻移至 R12 或 R24 处，以将 GPIO32 和 GPIO33 的连接从晶振移至 JP1。
### SPI Flash / JP2

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLK / GPIO6</td>
</tr>
<tr>
<td>2</td>
<td>SD0 / GPIO7</td>
</tr>
<tr>
<td>3</td>
<td>SD1 / GPIO8</td>
</tr>
<tr>
<td>4</td>
<td>SD2 / GPIO9</td>
</tr>
<tr>
<td>5</td>
<td>SD3 / GPIO10</td>
</tr>
<tr>
<td>6</td>
<td>CMD / GPIO11</td>
</tr>
</tbody>
</table>

**重要：**模组的 flash 总线已通过 0 欧电阻 R140 ~ R145 连接至排针 JP2。如果需要将 flash 的工作频率控制在 80 MHz，以达到保证总线信号完整性等目的，建议移除 R140 ~ R145 电阻，将模组的 flash 总线与排针 JP2 断开。

### JTAG / JP2

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th>JTAG 信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EN</td>
</tr>
<tr>
<td>2</td>
<td>TRST_N</td>
</tr>
<tr>
<td>3</td>
<td>TMS</td>
</tr>
<tr>
<td>4</td>
<td>TDO</td>
</tr>
<tr>
<td>5</td>
<td>TDI</td>
</tr>
<tr>
<td>6</td>
<td>TCK</td>
</tr>
</tbody>
</table>

### 摄像头 / JP4

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th>摄像头信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n/a</td>
</tr>
<tr>
<td>2</td>
<td>3.3V</td>
</tr>
<tr>
<td>3</td>
<td>地</td>
</tr>
<tr>
<td>4</td>
<td>SIO_C / SCCB 时钟</td>
</tr>
<tr>
<td>5</td>
<td>SIO_D / SCCB 数据</td>
</tr>
<tr>
<td>6</td>
<td>VSYNC/ 垂直同步</td>
</tr>
<tr>
<td>7</td>
<td>GPIO23</td>
</tr>
<tr>
<td>8</td>
<td>PCLK / 像素时钟</td>
</tr>
<tr>
<td>9</td>
<td>XCLK / 系统时钟</td>
</tr>
<tr>
<td>10</td>
<td>GPIO35</td>
</tr>
<tr>
<td>11</td>
<td>D7 / 像素数据 Bit 7</td>
</tr>
<tr>
<td>12</td>
<td>GPIO34</td>
</tr>
<tr>
<td>13</td>
<td>D6 / 像素数据 Bit 6</td>
</tr>
<tr>
<td>14</td>
<td>GPIO39</td>
</tr>
<tr>
<td>15</td>
<td>D5 / 像素数据 Bit 5</td>
</tr>
<tr>
<td>16</td>
<td>GPIO36</td>
</tr>
<tr>
<td>17</td>
<td>D4 / 像素数据 Bit 4</td>
</tr>
<tr>
<td>18</td>
<td>GPIO19</td>
</tr>
<tr>
<td>19</td>
<td>D3 / 像素数据 Bit 3</td>
</tr>
<tr>
<td>20</td>
<td>GPIO18</td>
</tr>
<tr>
<td>21</td>
<td>D2 / 像素数据 Bit 2</td>
</tr>
<tr>
<td>22</td>
<td>GPIO5</td>
</tr>
<tr>
<td>23</td>
<td>D1 / 像素数据 Bit 1</td>
</tr>
<tr>
<td>24</td>
<td>GPIO4</td>
</tr>
<tr>
<td>25</td>
<td>D0 / 像素数据 Bit 0</td>
</tr>
<tr>
<td>26</td>
<td>GPIO00</td>
</tr>
<tr>
<td>27</td>
<td>RESET / 摄像头复位</td>
</tr>
<tr>
<td>28</td>
<td>n/a</td>
</tr>
<tr>
<td>29</td>
<td>PWDN / 摄像头断电</td>
</tr>
</tbody>
</table>

- D0 到 D7 为摄像头的数据总线

### RGB LED

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th>RGB LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO0</td>
</tr>
<tr>
<td>2</td>
<td>红色</td>
</tr>
<tr>
<td>3</td>
<td>GPIO2</td>
</tr>
<tr>
<td>3</td>
<td>绿色</td>
</tr>
<tr>
<td>3</td>
<td>GPIO4</td>
</tr>
<tr>
<td>3</td>
<td>蓝色</td>
</tr>
</tbody>
</table>

microSD 卡

---

Espressif Systems

Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
章節1. 快速入門

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管腳</th>
<th>microSD 信號</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>MTDI / GPIO12</td>
<td>DATA2</td>
</tr>
<tr>
<td>2.</td>
<td>MTCK / GPIO13</td>
<td>CD / DATA3</td>
</tr>
<tr>
<td>3.</td>
<td>MTDO / GPIO15</td>
<td>CMD</td>
</tr>
<tr>
<td>4.</td>
<td>MTMS / GPIO14</td>
<td>CLK</td>
</tr>
<tr>
<td>5.</td>
<td>GPIO2</td>
<td>DATA0</td>
</tr>
<tr>
<td>6.</td>
<td>GPIO4</td>
<td>DATA1</td>
</tr>
<tr>
<td>7.</td>
<td>GPIO21</td>
<td>Card Detect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LCD / U5</th>
<th></th>
<th>ESP32 管腳</th>
<th>LCD 信號</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO18</td>
<td>复位</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>GPIO19</td>
<td>SCL</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>GPIO21</td>
<td>D/C</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>GPIO22</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GPIO23</td>
<td>SDA</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>GPIO25</td>
<td>SDO</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>GPIO5</td>
<td>背光</td>
<td></td>
</tr>
</tbody>
</table>

**應用程序开发** ESP-WROVER-KIT 上电前，请首先确认开发板完好无损。

**初始设置** 请严格按照下图所示连接跳线帽，注意不要额外连接其他跳线帽。

- 使用 JP7 连接器，选择 USB 为开发板供电。
- 使用 JP2 连接器，使能 UART 通信。

<table>
<thead>
<tr>
<th>USB 供电</th>
<th>使能 UART 通信</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="USB 供电" /></td>
<td><img src="image2" alt="使能 UART 通信" /></td>
</tr>
</tbody>
</table>
注意不要连接其他跳线帽。
打开 电源开关，5V Power On LED 应点亮。

正式开始开发 现在，请前往 快速入门 中的 安装 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

您可以在 IDF 组件注册器 中下载板级支持包 (BSP)。

以下链接提供了与 ESP-WROVER-KIT 开发板硬件相关的示例:

- 板上 LCD 示例：peripherals/spi_master/lcd
- SD 卡槽示例：storage/sd_card
- 摄像头示例：https://github.com/espressif/esp32-camera

相关文档

- ESP-WROVER-KIT V4.1 原理图 (PDF)
- ESP-WROVER-KIT V4.1 布局 (DXF)
- 《ESP32 技术规格书》 (PDF)
- 《ESP32-WROVER-E 技术规格书》 (PDF)
- JTAG 调试
- H/W 硬件参考

ESP-WROVER-KIT V3 入门指南

本指南介绍了如何开始使用 ESP-WROVER-KIT V3 开发板及其功能和相关配置。有关 ESP-WROVER-KIT 其他版本的介绍，请见：H/W 硬件参考。

准备工作

- ESP-WROVER-KIT V3 开发板
- USB 数据线（A 转 Micro-B）
- PC（Windows、Linux 或 macOS）

您可以跳过介绍部分，直接前往 应用程序开发 章节。

概述 ESP-WROVER-KIT 是 乐鑫 一款基于 ESP32 的开发板，集成了 LCD 屏幕和 microSD 卡槽。

ESP-WROVER-KIT 可选贴以下 ESP32 模组:

- ESP32-WROOM32
- ESP32-WROVER 系列模组

此外，ESP-WROVER-KIT 的独特之处在于集成了一款先进多协议 USB 桥接器 (FTDI FT2232HL)，允许开发人员直接通过 USB 接口，使用 JTAG 对 ESP32 进行调试，无需额外的 JTAG 调试器。ESP-WROVER-KIT 可为开发人员提供简单、便捷且极具成本效益的开发体验。

为了便于使用，板上模组的绝大部分管脚均已引出至开发板的引脚。

备注：该版本 ESP32-WROVER 模组的 GPIO16 和 GPIO17 管脚用作 PSRAM 的片选和时钟信号。默认情况下，为了给用户提供可靠的性能，这两个 GPIO 管脚不引出至开发板引脚。

功能概述 ESP-WROVER-KIT 开发板的主要组件和连接方式如下图所示。
图 9: ESP-WROVER-KIT 功能框图

功能说明 ESP-WROVER-KIT 开发板的主要组件、接口及控制方式见下。
下表介绍了开发板的主要组件，顺序如下：
- 从图片右上角开始，以顺时针顺序介绍了图 1 中的主要组件
- 然后以同样的顺序介绍了图 2 中的主要组件
图10: ESP-WROVER-KIT 开发板布局-俯视图
图 11: ESP-WROVER-KIT 开发板布局-仰视图
### 主要组件

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.768 kHz</td>
<td>外接 32.768 kHz 晶振，可提供 Deep-sleep 下使用的低功耗时钟。</td>
</tr>
<tr>
<td>0 欧电阻</td>
<td>ESP-WROVER-KIT 开发板设计了一个 0 欧电阻，可在测量 ESP32 系列模组在不同功耗模式下的电流时，直接移除或替换为分流器。</td>
</tr>
<tr>
<td>ESP32 模组</td>
<td>可选配 ESP32-WROOM-32 或 ESP32-WROOM, ESP32-WROVER 模组完整集成了 ESP32-WROOM-32 的所有功能，且内置 32-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。</td>
</tr>
<tr>
<td>FT2232</td>
<td>FT2232 多协议 USB 转串口桥接器。开发人员可通过 USB 接口对 FT2232 芯片进行控制和编程。ESP32 建立连接。FT2232 芯片可在通道 A 提供 USB-to-JTAG 接口功能，并在通道 B 提供 USB-to-Serial 接口功能。FT2232 使用 ESP32 模组串口，则可移除相关跳线帽，将模组连接至其他外部串口设备。</td>
</tr>
<tr>
<td>UART</td>
<td>串行 FT2232HL 和 ESP32 的串行 TX/RX 信号已引出至 JP11 的两端。默认情况下，这两路信号由跳线帽连接。如果要跳过 FT2232 使用 ESP32 模组串口，则可移除相关跳线帽，将模组连接至其他外部串口设备。</td>
</tr>
<tr>
<td>SPI</td>
<td>默认情况下，ESP32 使用 SPI 接口访问内置 flash 和 PSRAM，使用这些引脚连接 ESP32 和其他 SPI 设备。这种情况下，需增加额外的片选 (CS) 信号。注意，选配 ESP32-WROVER 模组时，该接口的工作电压为 1.8 V；选配 ESP32-WROOM-32 时，该接口的工作电压为 3.3 V。</td>
</tr>
<tr>
<td>CTS/RTS</td>
<td>串口流控信号，管脚默认不连接至电路。为了使能该功能，必须用跳线帽短路掉 JP14 的相应管脚。</td>
</tr>
<tr>
<td>JTAG</td>
<td>JTAG 接口，FT2232HL 和 ESP32 的 JTAG 信号已引出至 JP8 的两端。默认情况下，这两路信号不连接。如需使能 JTAG，请按照设置选项 的介绍，连接跳线帽。</td>
</tr>
<tr>
<td>EN</td>
<td>复位按键。</td>
</tr>
<tr>
<td>Boot</td>
<td>下载按键。按下 Boot 键并保持，同时按下 EN 键（此时不要松开 Boot 键）进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>USB</td>
<td>USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。</td>
</tr>
<tr>
<td>电源开关</td>
<td>按住 USB 键并保持，开发板上电；按住 USB 键并保持，开发板掉电。</td>
</tr>
<tr>
<td>电源选择开关</td>
<td>ESP-WROVER-KIT 开发板可通过 USB 端口或 5V 输入接口供电。用户可用跳线帽在两种供电模式中进行选择。更多详细信息，请见章节设置选项 中有关 JP7 连接器的描述。</td>
</tr>
<tr>
<td>5V Input</td>
<td>5V 电源接口建议仅在开发板自动运行（未连接 PC）时使用。仅用于全负荷工作下的后备电源。</td>
</tr>
<tr>
<td>LDO</td>
<td>5V-3.3V 低压差线型稳压器 NCP1117(1A)，NCP1117 最大电流输出为 1A。板上 LDO 为固定输出电压，但用户也可以选用具有可变输出电压的 LDO。更多信息，请见 ESP-WROVER-KIT V3 原型图。</td>
</tr>
<tr>
<td>摄像头</td>
<td>摄像头接口，支持标准 OV7670 摄像头模块。</td>
</tr>
<tr>
<td>RGB LED</td>
<td>红绿蓝发光二极管，可由 PWM（脉冲宽度调制）控制。</td>
</tr>
<tr>
<td>I/O</td>
<td>板上模组的所有管脚均引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。</td>
</tr>
<tr>
<td>microSD 卡槽</td>
<td>适用于需要扩充数据存储空间或进行备份的应用开发场景。</td>
</tr>
<tr>
<td>LCD 显示屏</td>
<td>支持贴装一款 3.2” SP (标准四线串行外设接口) LCD 显示器，请见 ESP-WROVER-KIT 开发板布局及视图。</td>
</tr>
</tbody>
</table>

### 设置选项

用户可通过 5 组排针，设置开发板功能，其中常见功能见下表：

---

**注意**：以上信息可能因版本更新而有所变化。详情请参阅官方文档。
<table>
<thead>
<tr>
<th>排针</th>
<th>线路设置</th>
<th>功能描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP7</td>
<td></td>
<td>使用外部电源为 ESP-WROVER-KIT 开发板供电</td>
</tr>
<tr>
<td>JP7</td>
<td></td>
<td>使用 USB 端口为 ESP-WROVER-KIT 开发板供电</td>
</tr>
<tr>
<td>JP8</td>
<td></td>
<td>使能 JTAG 功能</td>
</tr>
</tbody>
</table>
**ESP32 管脚分配**

ESP32 模组的部分管脚/端口已被板上组件占用或用于外部硬件设备。如果某管脚对应的特定硬件未连接，则该管脚可作他用。比如，摄像头/JP4 排针未连接对应硬件，则这些 GPIO 可用于其他用途。

部分管脚具备多个功能，可供板上组件或外部硬件设备同时使用，比如 GPIO0 和 GPIO2。由于管脚限制，一些外围设备不可同时使用，比如，由于 JTAG 和 SD 卡槽需共用部分管脚，因此一些使用 SD 卡功能的应用无法同时进行 JTAG 调试。

其他情况下，不同外设可同时使用。比如，LCD 屏幕和 SD 卡仅用一个 GPIO21 管脚，可以同时使用。该管脚可为 LCD 屏幕提供 D/C（数据/控制）信号，并用于读取来自 SD 卡槽的 CD 信号（卡检测信号）。如无需使用卡检测功能，开发人员还可以通过移除 R167 来禁用该功能。此时，LCD 和 SD 卡槽可同时使用。

更多外设共享管脚的介绍，请见下章节中的表格。

**主要 I/O 连接器 / JP1**

JP1 连接器包括 14 x 2 个排针，具体功能可见下表中间 “I/O” 列的介绍。两侧的“共用”列则介绍了这些管脚在板上的其他用途。

<table>
<thead>
<tr>
<th>共用</th>
<th>I/O</th>
<th>I/O</th>
<th>共用</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>3.3V</td>
<td>GND</td>
<td>n/a</td>
</tr>
<tr>
<td>NC/XTAL</td>
<td>IO32</td>
<td>IO33</td>
<td>NC/XTAL</td>
</tr>
<tr>
<td>JTAG, microSD</td>
<td>IO12</td>
<td>IO13</td>
<td>JTAG, microSD</td>
</tr>
<tr>
<td>JTAG, microSD</td>
<td>IO14</td>
<td>IO27</td>
<td>摄像头</td>
</tr>
<tr>
<td>摄像头</td>
<td>IO26</td>
<td>IO25</td>
<td>摄像头, LCD</td>
</tr>
<tr>
<td>摄像头</td>
<td>IO35</td>
<td>IO34</td>
<td>摄像头</td>
</tr>
<tr>
<td>摄像头</td>
<td>IO39</td>
<td>IO36</td>
<td>摄像头</td>
</tr>
<tr>
<td>JTAG</td>
<td>EN</td>
<td>IO23</td>
<td>摄像头, LCD</td>
</tr>
<tr>
<td>摄像头, LCD</td>
<td>IO22</td>
<td>IO21</td>
<td>摄像头, LCD, microSD</td>
</tr>
<tr>
<td>摄像头, LCD</td>
<td>IO19</td>
<td>IO18</td>
<td>摄像头, LCD</td>
</tr>
<tr>
<td>摄像头, LCD</td>
<td>IO5</td>
<td>IO17</td>
<td>PSRAM</td>
</tr>
<tr>
<td>PSRAM</td>
<td>IO16</td>
<td>IO4</td>
<td>LED, 摄像头, microSD</td>
</tr>
<tr>
<td>摄像头, LED, Boot</td>
<td>IO0</td>
<td>IO2</td>
<td>LED, microSD</td>
</tr>
<tr>
<td>JTAG, microSD</td>
<td>IO15</td>
<td>5V</td>
<td></td>
</tr>
</tbody>
</table>

说明：
- NC/XTAL - 32.768 kHz Oscillator
- JTAG - JTAG / JP8
- Boot - Boot 按键 / SW2
- 摄像头 - 摄像头 / JP4
- LED - RGB LED
- microSD - microSD Card / J4
- LCD - LCD / U5
- PSRAM - 仅适用于选装 ESP32-WROVER 的情况。

**32.768 kHz 晶振**

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>GPIO32</td>
</tr>
<tr>
<td>2.</td>
<td>GPIO33</td>
</tr>
</tbody>
</table>

备注：默认情况下，管脚 GPIO32 和 GPIO33 已连接至晶振。因此，为了保证信号的完整性，这两个管脚并未连接至 JP1 I/O 连接器。用户可通过将 R11/R23 处的 0 欧电阻移至 R12/R24 处，以将 GPIO32 和...
GPIO33 的连接从晶振移至 JP1。

### SPI Flash / JP13

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>CLK / GPIO6</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>SD0 / GPIO7</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>SD1 / GPIO8</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>SD2 / GPIO9</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>SD3 / GPIO10</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>CMD / GPIO11</td>
<td></td>
</tr>
</tbody>
</table>

**重要：** 模组的 flash 总线已通过 0 欧电阻 R140 ~ R145 连接至排针 JP13。如果需要将 flash 的工作频率控制在 80 MHz，为了达到保证总线信号完整性等目的，建议移除 R140 ~ R145 电阻。将模组的 flash 总线与排针 JP2 断开。

### JTAG / JP8

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
<th>JTAG 信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>EN</td>
<td>TRST_N</td>
</tr>
<tr>
<td>2.</td>
<td>MTMS / GPIO14</td>
<td>TMS</td>
</tr>
<tr>
<td>3.</td>
<td>MTDO / GPIO15</td>
<td>TDO</td>
</tr>
<tr>
<td>4.</td>
<td>MTDI / GPIO12</td>
<td>TDI</td>
</tr>
<tr>
<td>5.</td>
<td>MTCK / GPIO13</td>
<td>TCK</td>
</tr>
</tbody>
</table>

### 摄像头 / JP4
<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
<th>摄像头信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>n/a</td>
<td>3.3V</td>
</tr>
<tr>
<td>2.</td>
<td>n/a</td>
<td>地</td>
</tr>
<tr>
<td>3.</td>
<td>GPIO27</td>
<td>SIO_C / SCCB 时钟</td>
</tr>
<tr>
<td>4.</td>
<td>GPIO26</td>
<td>SIO_D / SCCB 数据</td>
</tr>
<tr>
<td>5.</td>
<td>GPIO25</td>
<td>VSYNC / 垂直同步</td>
</tr>
<tr>
<td>6.</td>
<td>GPIO23</td>
<td>HREF / 水平参考</td>
</tr>
<tr>
<td>7.</td>
<td>GPIO22</td>
<td>PCLK / 像素时钟</td>
</tr>
<tr>
<td>8.</td>
<td>GPIO21</td>
<td>XCLK / 系统时钟</td>
</tr>
<tr>
<td>9.</td>
<td>GPIO35</td>
<td>D7 / 像素数据 Bit 7</td>
</tr>
<tr>
<td>10.</td>
<td>GPIO34</td>
<td>D6 / 像素数据 Bit 6</td>
</tr>
<tr>
<td>11.</td>
<td>GPIO39</td>
<td>D5 / 像素数据 Bit 5</td>
</tr>
<tr>
<td>12.</td>
<td>GPIO36</td>
<td>D4 / 像素数据 Bit 4</td>
</tr>
<tr>
<td>13.</td>
<td>GPIO19</td>
<td>D3 / 像素数据 Bit 3</td>
</tr>
<tr>
<td>14.</td>
<td>GPIO18</td>
<td>D2 / 像素数据 Bit 2</td>
</tr>
<tr>
<td>15.</td>
<td>GPIO5</td>
<td>D1 / 像素数据 Bit 1</td>
</tr>
<tr>
<td>16.</td>
<td>GPIO4</td>
<td>D0 / 像素数据 Bit 0</td>
</tr>
<tr>
<td>17.</td>
<td>GPIO00</td>
<td>RESET / 摄像头复位</td>
</tr>
<tr>
<td>18.</td>
<td>n/a</td>
<td>PWDN / 摄像头断电</td>
</tr>
</tbody>
</table>

* D0 到 D7 为摄像头的数据总线

**RGB LED**
### ESP32 管脚

<table>
<thead>
<tr>
<th>GPIO</th>
<th>RGB LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>GPIO0</td>
</tr>
<tr>
<td>2.</td>
<td>GPIO2</td>
</tr>
<tr>
<td>3.</td>
<td>GPIO4</td>
</tr>
</tbody>
</table>

### microSD 卡

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th>microSD 信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. MTDI / GPIO12</td>
<td>DATA2</td>
</tr>
<tr>
<td>2. MTCK / GPIO13</td>
<td>CD / DATA3</td>
</tr>
<tr>
<td>3. MTDO / GPIO15</td>
<td>CMD</td>
</tr>
<tr>
<td>4. MTMS / GPIO14</td>
<td>CLK</td>
</tr>
<tr>
<td>5. GPIO2</td>
<td>DATA0</td>
</tr>
<tr>
<td>6. GPIO4</td>
<td>DATA1</td>
</tr>
<tr>
<td>7. GPIO21</td>
<td>CD</td>
</tr>
</tbody>
</table>

### LCD / U5

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th>LCD 信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GPIO18</td>
<td>复位</td>
</tr>
<tr>
<td>2. GPIO19</td>
<td>SCL</td>
</tr>
<tr>
<td>3. GPIO21</td>
<td>D/C</td>
</tr>
<tr>
<td>4. GPIO22</td>
<td>CS</td>
</tr>
<tr>
<td>5. GPIO23</td>
<td>SDA</td>
</tr>
<tr>
<td>6. GPIO25</td>
<td>SDO</td>
</tr>
<tr>
<td>7. GPIO5</td>
<td>背光</td>
</tr>
</tbody>
</table>

应用程序开发：ESP-WROVER-KIT 上电前，请首先确认开发板完好无损。
初始设置 请严格按照下图所示连接跳线帽，注意不要额外连接其他跳线帽。

- 使用 J7 连接器，选择 USB 为开发板供电。
- 使用 J11 连接器，使能 UART 通信。

<table>
<thead>
<tr>
<th>USB 供电</th>
<th>使能 UART 通信</th>
</tr>
</thead>
</table>

注意不要连接其他跳线帽。
打开 电源开关，5V Power On LED 应点亮。

正式开始开发 现在，请前往快速入门 中的安装 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

相关文档
- ESP-WROVER-KIT V3 原理图 (PDF)
- 《ESP32 技术规格书》 (PDF)
- 《ESP32-WROVER 技术规格书》 (PDF)
- 《ESP32-WROOM-32 技术规格书》 (PDF)
- JTAG 调试
- H/W 硬件参考

ESP-WROVER-KIT V2 入门指南
本指南介绍了如何开始使用 ESP-WROVER-KIT V2 开发板及其功能和相关配置。有关 ESP-WROVER-KIT 其他版本的介绍，请见 H/W 硬件参考。

准备工作
- ESP-WROVER-KIT V2 开发板
- USB 数据线（A 转 Micro-B）
- PC（Windows，Linux 或 macOS）

您可以跳过介绍部分，直接前往应用程序开发 章节。

概述  ESP-WROVER-KIT 是 乐鑫 一款基于 ESP32 的开发板，集成了 LCD 屏幕和 microSD 卡槽。
ESP-WROVER-KIT 可选配以下 ESP32 模组：
- ESP32-WROOM-32
• ESP32-WROVER 系列模组
此外, ESP-WROVER-KIT 的独特之处在于集成了一款先进多协议 USB 桥接器 (FTDI FT2232HL), 允许开发人员直接通过 USB 接口, 使用 JTAG 对 ESP32 进行调试, 无需额外的 JTAG 调试器。ESP-WROVER-KIT 可为开发人员提供简单、便捷且极具成本效益的开发体验。
为了便于使用, 板上模组的绝大部分管脚均已引出至开发板的引脚。

备注: ESP-WROVER-KIT V2 板载 ESP32-WROVER 模组的 GPIO16 和 GPIO17 管脚用作 PSRAM 的片选和时钟信号。默认情况下, 为了给用户提供可靠的性能, 这两个 GPIO 管脚不引出至开发板引脚。

功能概述 ESP-WROVER-KIT 开发板的主要组件和连接方式如下图所示。

图 12: ESP-WROVER-KIT 功能框图

功能说明 ESP-WROVER-KIT 开发板的主要组件、接口及控制方式见下。
下表介绍了开发板的主要组件，顺序如下:
• 从图片右上角开始，以顺时针顺序介绍了图 1 中的主要组件
• 然后以同样的顺序介绍了图 2 中的主要组件
图 13: ESP-WROVER-KIT 开发板布局—俯视图
图 14: ESP-WROVER-KIT 开发板布局—仰视图
主要组件 | 基本介绍
---|---
32.768 kHz | 外接 32.768 kHz 振荡器，可提供 Deep-sleep 下使用的低功耗时钟。
ESP32 模组 | 可选配 ESP32-WROOM-32 或 ESP32-WROVER。ESP32-WROVER 模组集合了 ESP32-WROOM-32 的所有功能，且内置 32-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。
CTS/RTS | 串口流控信号。管脚默认不连接至电路。为了使能该功能，必须用跳线帽短路掉 JP14 的相应管脚。
UART | 串口。FT2232HL 和 ESP32 的串行 TX/RX 信号已引出至 JP11 的两端。默认情况下，这两路信号由跳线帽连接。如果要跳过 FT2232 使用 ESP32 模组串口，则可移除相关跳线帽，将模组连接至其他外部串口设备。
SPI | 默认情况下，ESP32 使用 SPI 接口访问内核 flash 和 PSRAM。使用这些引脚连接 ESP32 和其他 SPI 设备。这种情况下，需增加额外的片选 (CS) 信号。注意，选配 ESP32-WROVER 模组时，该接口的工作电压为 1.8 V；选配 ESP32-WROOM-32 时，该接口的工作电压为 3.3 V。
JTAG | JTAG 接口，FT2232HL 和 ESP32 的 JTAG 信号已引出至 JP8 的两端。默认情况下，这两路信号不连接。如需使能 JTAG，请按照设置选项的介绍，连接跳线帽。
FT2232 | FT2232 多协议 USB 转串口桥接器。开发者可通过 USB 接口对 FT2232 芯片进行控制和编程，与 ESP32 建立连接。FT2232 具有 USB-to-uart 和 USB-to-JTAG 功能。

EN | 复位按键。
Boot | 下载按键。按下 Boot 键并保持，同时按一下 EN 键（此时不要松开 Boot 键）进入“固件下载”模式，通过串口下载固件。
USB | USB 接口，可用作开发板的供电电源，或连接 PC 和开发板的通信接口。
电源选择开关 | ESP-WROVER-KIT 开发板可通过 USB 端口或 5 V 输入接口供电。用户可使用跳线帽在两种供电模式中进行选择。更多详细信息，请见章节设置选项中有关 JP7 连接器的描述。
电源开关 | 拨向 USB 按键一侧，开发板上电；拨向 USB 按键一侧，开发板掉电。
5V Input | 5 V 电源接口建议仅在开发板自动运行（未连接 PC）时使用，仅用于全负荷工作下的后备电源。
LDO | 5V-to-3.3V 低电压差线型稳压器 NCP1117(1A)，NCP1117 最大电流输出为 1 A。板上 LDO 为固定输出电压，但用户也可以选用具有可变输出电压的 LDO。更多信息，请见 ESP-WROVER-KIT V2 原理图。
摄像头 | 摄像头接口，支持标准 OV7670 摄像头模块。
RGB LED | 红绿蓝发光二极管，可由 PWM（脉冲宽度调制）控制。
I/O | 板上模组的所有管脚均已引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。
microSD 卡槽 | microSD 卡槽，可扩充存储空间：当 ESP32 进入下载模式时，GPIO2 不可处于高电平。然而，为了使能 microSD 卡功能，需为 GPIO2 增加一个上拉电阻。默认情况下，GPIO2 和上拉电阻 R153 处于断开状态。为了使能 microSD 卡，请按照设置选项章节的要求，连接 JP11 连接器。
LCD 显示屏 | 支持贴装一款 3.2 " SPI (标准四线串行外设接口) LCD 显示器。请见 ESP-WROVER-KIT 开发板布局—俯视图。

设置选项 | 用户可通过 5 组排针，设置开发板功能，其中常见功能见下表：

设置选项    | 功能描述
---|---
用户可通过 5 组排针，设置开发板功能，其中常见功能见下表：
<table>
<thead>
<tr>
<th>针号</th>
<th>跳线设置</th>
<th>功能描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP1</td>
<td><img src="image1.png" alt="JP1-img" /></td>
<td>使能 microSD Card 功能的上拉电阻</td>
</tr>
<tr>
<td>JP1</td>
<td><img src="image2.png" alt="JP1-img" /></td>
<td>确保开发板处于下载模式时，GPIO2 时钟处于低位（将 JP1 连接至 GPIO0）</td>
</tr>
<tr>
<td>JP7</td>
<td><img src="image3.png" alt="JP7-img" /></td>
<td>使用外部电源为 ESP-WROVER-KIT 开发板供电</td>
</tr>
<tr>
<td>JP7</td>
<td><img src="image4.png" alt="JP7-img" /></td>
<td>使用 USB 端口为 ESP-WROVER-KIT 开发板供电</td>
</tr>
<tr>
<td>JP8</td>
<td><img src="image5.png" alt="JP8-img" /></td>
<td>使能 JTAG 功能</td>
</tr>
</tbody>
</table>
应用程序开发  ESP-WROVER-KIT 上电前，请首先确认开发板完好无损。

初始设置  请严格按照图示连接跳线帽。注意不要额外连接其他跳线帽。

- 使用 JP7 连接器，选择 USB 为开发板供电。
- 使用 JP11 连接器，使能 UART 通信。

注意不要连接其他跳线帽。

打开 电源开关，5V Power On LED 应点亮。

正式开始开发  现在，请前往快速入门中的 安装 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

相关文档
- ESP-WROVER-KIT V2 原理图 (PDF)
- 《ESP32 技术规格书》 (PDF)
- 《ESP32-WROVER 技术规格书》 (PDF)
- 《ESP32-WROOM-32 技术规格书》 (PDF)
- JTAG 调试
- H/W 硬件参考

ESP32-PICO-KIT V4/V4.1 入门指南

本指南介绍了如何开始使用 ESP32-PICO-KIT V4/V4.1 迷你开发板。有关 ESP32-PICO-KIT 其他版本的介绍，请见：H/W 硬件参考。

本指南仅适用于 ESP32-PICO-KIT V4 和 V4.1。ESP32-PICO-KIT V4.1 与 V4 的最大差别在于桥接器。其中 V4 搭载的 CP2102 USB-to-UART 桥接器最高速率为 1 Mbps，V4.1 搭载的 CP2102N 桥接器最高传输速率为 3 Mbps。

准备工作
- ESP32-PICO-KIT 迷你开发板
- USB 2.0 线（A 型转 Micro-B 型）
- PC（Windows、Linux 或 macOS）

您可以跳过介绍部分，直接前往应用程序开发章节。
概要  ESP32-PICO-KIT 是一款来自 乐鑫 的迷你开发板。

该开发板的核心是具有完整 Wi-Fi 和蓝牙功能的 ESP32 系列 SiP 模组 ESP32-PICO-D4。与其他 ESP32 系列模组相比，ESP32-PICO-D4 模组已集成以下外围器件:

- 40 MHz 晶体振荡器
- 4 MB flash
- 滤波电容
- 射频匹配网络等

这大大降低了用户额外采购和安装这些元件件的数量和成本，及额外组装测试的复杂度，并增加了可用性。

ESP32-PICO-KIT 集成了 USB 转 UART 桥接电路，允许开发人员直接通过 PC 的 USB 端口进行下载和调试。

为了便于连接，ESP32-PICO-D4 上的所有 IO 信号和系统电源管脚均通过开发板两侧焊盘（每侧 20 个 x 0.1 英寸间隔）引出。为了方便板间线的使用，ESP32-PICO-KIT 开发板每侧的 20 个焊盘中，有 17 个引出至排针，另外 3 个靠近天线的焊盘未引出，可供用户日后焊接使用。

备注:
1. ESP32-PICO-D4 开发板目前有两种版本，分别采用排针与排母。本指南默认以排针版本为例。
2. 每排未引出至排针的 3 个管脚已连接至 ESP32-PICO-D4 SiP 模组的内置 flash 模块。更多信息，请见相关文档 中的模组技术规格书。

功能概要  ESP32-PICO-KIT 开发板的主要组件和连接方式见下。

功能说明  ESP32-PICO-KIT 开发板的主要组件、接口及控制方式见下。ESP32-PICO-KIT 开发板的主要组件描述见下表（从左上角起顺时针顺序）。

图 15: ESP32-PICO-KIT 框图
图 16: ESP32-PICO-KIT 开发板布局（排母版）

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-PICO-D4</td>
<td>ESP32-PICO-KIT 开发板上焊接的标准 ESP32-PICO-D4 模组，集成了 ESP32 芯片的完整系统，仅需连接天线、LC 匹配电路、退耦电容和一个 EN 信号上拉电阻即可正常工作。</td>
</tr>
<tr>
<td>LDO</td>
<td>5V-to-3.3V 低压差稳压器</td>
</tr>
<tr>
<td>USB-to-UART 桥接器</td>
<td>单芯片 USB-to-UART 桥接器。V4 版本搭载的 CP2102 可提供高达 1 Mbps 的传输速率。V4.1 版本搭载的 CP2102N 可提供高达 3 Mbps 的传输速率。</td>
</tr>
<tr>
<td>Micro USB 端口</td>
<td>USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。</td>
</tr>
<tr>
<td>5V Power On LED</td>
<td>开发板通电后，该红色指示灯将亮起。更多信息，请见相关文档中的原理图。</td>
</tr>
<tr>
<td>I/O</td>
<td>ESP32-PICO-D4 的所有管脚均已引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。更多详情，请见章节管脚说明。</td>
</tr>
<tr>
<td>BOOT</td>
<td>下载按键。按下 Boot 键并保持，同时按一下 EN 键（此时不要松开 Boot 键）进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>EN</td>
<td>复位按键。</td>
</tr>
</tbody>
</table>

电源选项： 开发板可任选一选用以下三种供电方式：
- Micro USB 供电（默认）
- 5V / GND 管脚供电
- 3V3 / GND 管脚供电

警告：上述供电模式不可同时连接，否则可能会损坏开发板和/或电源。

管脚说明： 下表介绍了开发板 I/O 管脚的名称和功能，具体布局请见相关文档中的原理图。请参考 ESP32-PICO-KIT 开发板布局（排母版）。
<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FLASH_SD1 (FSD1)</td>
<td>I/O</td>
<td>GPIO8, SD_DATA1, SPID, HS1_DATA1 (见说明 1), U2CTS</td>
</tr>
<tr>
<td>2</td>
<td>FLASH_SD3 (FSD3)</td>
<td>I/O</td>
<td>GPIO7, SD_DATA0, SPIQ, HS1_DATA0 (见说明 1), U2RTS</td>
</tr>
<tr>
<td>3</td>
<td>FLASH_CLK (FCLK)</td>
<td>I/O</td>
<td>GPIO6, SD_CLK, SPICLK, HS1_CLK (见说明 1), U1CTS</td>
</tr>
<tr>
<td>4</td>
<td>IO21</td>
<td>I/O</td>
<td>GPIO21, VSPIHD, EMAC_TX_EN</td>
</tr>
<tr>
<td>5</td>
<td>IO22</td>
<td>I/O</td>
<td>GPIO22, VSPIWP, U0RTS, EMAC_TXD1</td>
</tr>
<tr>
<td>6</td>
<td>IO19</td>
<td>I/O</td>
<td>GPIO19, VSPIQ, U0CTS, EMAC_TXD0</td>
</tr>
<tr>
<td>7</td>
<td>IO23</td>
<td>I/O</td>
<td>GPIO23, VSPID, HS1_STROBE</td>
</tr>
<tr>
<td>8</td>
<td>IO18</td>
<td>I/O</td>
<td>GPIO18, VSPICLK, HS1_DATA7</td>
</tr>
<tr>
<td>9</td>
<td>IO5</td>
<td>I/O</td>
<td>GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK</td>
</tr>
<tr>
<td>10</td>
<td>IO10</td>
<td>I/O</td>
<td>GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD</td>
</tr>
<tr>
<td>11</td>
<td>IO9</td>
<td>I/O</td>
<td>GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD</td>
</tr>
<tr>
<td>12</td>
<td>RXD0</td>
<td>I/O</td>
<td>GPIO3, U0RXD (见说明 3), CLK_OUT2</td>
</tr>
</tbody>
</table>
Chapter 1. 快速入门
<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FLASH_CS (FCS)</td>
<td>I/O</td>
<td>GPIO16, HS1_DATA4 (见说明 1), U2RXD, EMAC_CLK_OUT</td>
</tr>
<tr>
<td>2</td>
<td>FLASH_SD0 (FSD0)</td>
<td>I/O</td>
<td>GPIO17, HS1_DATA5 (见说明 1), U2TXD, EMAC_CLK_OUT_180</td>
</tr>
<tr>
<td>3</td>
<td>FLASH_SD2 (FSD2)</td>
<td>I/O</td>
<td>GPIO11, SD_CMD, SPICS0, HS1_CMD (见说明 1), U1RTS</td>
</tr>
<tr>
<td>4</td>
<td>SENSOR_VP (FSVP)</td>
<td>I</td>
<td>GPIO36, ADC1_CH0, RTC_GPIO0</td>
</tr>
<tr>
<td>5</td>
<td>SENSOR_VN (FSVN)</td>
<td>I</td>
<td>GPIO39, ADC1_CH3, RTC_GPIO3</td>
</tr>
<tr>
<td>6</td>
<td>IO25</td>
<td>I/O</td>
<td>GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0</td>
</tr>
<tr>
<td>7</td>
<td>IO26</td>
<td>I/O</td>
<td>GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1</td>
</tr>
<tr>
<td>8</td>
<td>IO32</td>
<td>I/O</td>
<td>32K_XP (见说明 2a), ADC1_CH4, TOUCH9, RTC_GPIO9</td>
</tr>
<tr>
<td>9</td>
<td>IO33</td>
<td>I/O</td>
<td>32K_XN (见说明 2b), ADC1_CH5, TOUCH8, RTC_GPIO8</td>
</tr>
<tr>
<td>10</td>
<td>IO27</td>
<td>I/O</td>
<td>GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV</td>
</tr>
<tr>
<td>11</td>
<td>IO14</td>
<td>I/O</td>
<td>GPIO18, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK</td>
</tr>
<tr>
<td>12</td>
<td>IO12</td>
<td>I/O</td>
<td>GPIO20, ADC2_CH3, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_CLK, SD_CLK, EMAC_TXD2</td>
</tr>
<tr>
<td>13</td>
<td>IO13</td>
<td>I/O</td>
<td>GPIO21, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA0, SD_DATA0</td>
</tr>
<tr>
<td>14</td>
<td>IO15</td>
<td>I/O</td>
<td>GPIO22, ADC2_CH0, TOUCH3, RTC_GPIO13, MTDO, HSPICLK, HS2_DATA1, SD_DATA1</td>
</tr>
<tr>
<td>15</td>
<td>IO0</td>
<td>I/O</td>
<td>GPIO23, ADC2_CH1, TOUCH2, RTC_GPIO12, CLK_OUT1, EMAC_TX_CLK</td>
</tr>
<tr>
<td>16</td>
<td>VDD33 (3V3)</td>
<td>P</td>
<td>3.3V</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>P</td>
<td>Ground</td>
</tr>
<tr>
<td>18</td>
<td>EXT_5V (5V)</td>
<td>P</td>
<td>5V</td>
</tr>
</tbody>
</table>
备注:

1. 该管脚已连接至 ESP32-PICO-D4 的内置 flash 管脚。
2. 32.768 kHz 晶振：(a) 输入；(b) 输出。
3. 该管脚已连接至开发板的 USB 桥接器芯片。
4. ESP32-PICO-KIT 内置 SPI flash 的工作电压为 3.3 V。因此，strapping 管脚 MTDI 在复位过程中应保持低电平。如连接该管脚，请确保该管脚在复位中不要保持高电平。

图 17: ESP32-PICO-KIT 管脚布局（点击放大）

管脚布局

应用程序开发

ESP32-PICO-KIT 上电前，请首先确认开发板完好无损。

现在，请前往快速入门中的安装章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

开发板尺寸

ESP32-PICO-KIT 的尺寸为 52 x 20.3 x 10 mm (2.1” x 0.8” x 0.4” )。

图 18: ESP32-PICO-KIT 尺寸图—背面（排针版）

有关开发板的物理结构细节，请见下方参考设计。
图 19: ESP32-PICO-KIT 尺寸图-侧面（排针版）

相关文档
- ESP32-PICO-KIT V4 原理图 (PDF)
- ESP32-PICO-KIT V4.1 原理图 (PDF)
- ESP32-PICO-KIT 参考设计，内含 OrCAD 原理图、PCB 布局、Gerbers 和 BOM 表
- 《ESP32-PICO-D4 技术规格书》 (PDF)
- H/W 硬件参考

ESP32-PICO-KIT V3 入门指南
本指南介绍了如何开始使用 ESP32-PICO-KIT V3 迷你开发板。有关 ESP32-PICO-KIT 其他版本的介绍，请见：H/W 硬件参考。

准备工作
- ESP32-PICO-KIT V3 迷你开发板
- USB 2.0 线（A 型转 Micro-B 型）
- PC（Windows、Linux 或 macOS）

您可以跳过介绍部分，直接前往应用程序开发章节。

概述  ESP32-PICO-KIT V3 是一款来自 乐鑫 的迷你开发板，其核心是具有完整 Wi-Fi 和蓝牙功能的 ESP32 系列 SiP 模组 ESP32-PICO-D4。

ESP32-PICO-KIT 集成了 USB 转 UART 桥接电路，允许开发人员直接通过 PC 的 USB 端口进行下载和调试。

为了便于连接，ESP32-PICO-D4 上的所有 IO 信号和系统电源管脚均通过开发板两侧焊盘（每侧 20 个 x 0.1 英寸间隔）引出。

功能说明  ESP32-PICO-KIT V3 开发板的主要组件、接口及控制方式见下。

ESP32-PICO-KIT 开发板的主要组件描述见下表。
图 20: ESP32-PICO-KIT V3 开发板布局
### 主要组件

<table>
<thead>
<tr>
<th>ESP32-PICO-D4</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-PICO-KIT V3 开发板上焊接的标准 ESP32-PICO-D4 模组，集成了 ESP32 芯片的完整系统。仅需连接天线、LC 匹配电路、退耦电容和一个EN信号上拉电阻即可正常工作。</td>
<td></td>
</tr>
<tr>
<td>LDO</td>
<td>5V-to-3.3V 低电压差稳压器。</td>
</tr>
<tr>
<td>USB-to-UART 模块</td>
<td>单芯片 USB-to-UART 模块，可提供高达 1Mhz 的传输速率。</td>
</tr>
<tr>
<td>Micro USB 端口</td>
<td>USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。</td>
</tr>
<tr>
<td>Power On LED</td>
<td>开发板通电后，该红色指示灯将亮起。</td>
</tr>
<tr>
<td>I/O</td>
<td>ESP32-PICO-D4 的所有管脚均已引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。</td>
</tr>
<tr>
<td>BOOT</td>
<td>下载按键。按下 Boot 键并保持，同时按一下 EN 键（此时不要松开 Boot 键）进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>EN</td>
<td>复位按键。</td>
</tr>
</tbody>
</table>

### 应用程序开发

ESP32-PICO-KIT V3 上电前，请首先确认开发板完好无损。

现在，请前往快速入门中的安装章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

### 相关文档

- ESP32-PICO-KIT V3 原理图 (PDF)
- 《ESP32-PICO-D4 技术规格书》 (PDF)
- H/W 硬件参考

### ESP32-Ethernet-Kit V1.2 入门指南

本指南将介绍 ESP32-Ethernet-Kit 开发板的配置以及相关功能的使用。

**ESP32-Ethernet-Kit** 是一款以太网转 Wi-Fi 开发板，可为以太网设备赋予 Wi-Fi 连接功能。为了提供更灵活的电源选项，ESP32-Ethernet-Kit 同时也支持以太网供电 (PoE)。

### 准备工作

- **ESP32-Ethernet-Kit V1.2 开发板**
- USB 数据线（A 转 Micro-B）
- PC（Windows、Linux 或 macOS）

您可以跳过介绍部分，直接前往应用程序开发 章节。

### 概述

ESP32-Ethernet-Kit 是一款来自乐鑫的开发板。

它由以太网模块（A 板）和 PoE 子板（B 板）两部分组成。其中以太网模块（A 板）集成了蓝牙/Wi-Fi 双模 ESP32-WROVER-E 模组和单端口 10/100 Mbps 快速以太网收发器 (PHY) IP101GRI。PoE 子板（B 板）提供以太网供电功能。ESP32-Ethernet-Kit 的 A 板可在不连接 B 板的情况下独立工作。

为了实现程序下载和监控，A 板还集成了一款先进多协议 USB 桥接器（FT2232H 芯片）。FT2232H 芯片使得开发人员无需额外的 JTAG 适配器，通过 USB 桥接器使用 JTAG 接口便可对 ESP32 直接进行调试。

### 功能概述

ESP32-Ethernet-Kit 开发板的主要组件和连接方式如下。

### 功能说明

有关 ESP32-Ethernet-Kit 开发板的主要组件、接口及控制方式，请见下方的图片和表格。
图 21: ESP32-Ethernet-Kit V1.2 概图（点击放大）
图 22: ESP32-Ethernet-Kit V1.2（点击放大）

图 23: ESP32-Ethernet-Kit 功能框图（点击放大）
图 24: ESP32-Ethernet-Kit - 以太网母板（A 板）布局（点击放大）

以太网母板（A 板） 下表将从图片右上角开始，以顺时针顺序介绍图中的主要组件。
表1: 表格1 介绍

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-WROVER-E 模组</td>
<td>这款 ESP32 模组内置 64-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。</td>
</tr>
<tr>
<td>GPIO Header 2</td>
<td>由 5 个未引出通孔组成，可连接至 ESP32 的部分 GPIO。具体介绍，请见 GPIO Header 2。</td>
</tr>
<tr>
<td>功能选择开关</td>
<td>一个 4 位拨码开关，可配置 ESP32 部分 GPIO 的功能。具体介绍，请见 功能选择开关。</td>
</tr>
<tr>
<td>Tx/Rx LEDs</td>
<td>2 个 LED，可显示 UART 传输的状态。</td>
</tr>
<tr>
<td>FT2232H</td>
<td>FT2232H 多协议 USB 转串口模块通信。开发人员可与 FT2232H 芯片进行控制和编程，与 ESP32 建立连接。FT2232H 芯片可与以太网连接（如通过 USB-to-JTAG 接口），并在通过 B 提供 USB-to-Serial 接口功能，便利开发人员的开发与调试。见 ESP32-Ethernet-Kit V1.2 以太网母板（A 板）原理图。</td>
</tr>
<tr>
<td>USB 端口</td>
<td>USB 接口，可作为开发板的供电电源，或连接 PC 和开发板的通信接口。</td>
</tr>
<tr>
<td>电源开关</td>
<td>电源开关，同时控制 5V0 接线，开发板供电：按向 GND 键一侧，开发板掉电。</td>
</tr>
<tr>
<td>5V Input</td>
<td>5V 电源接口仅在开机自动运行 (未连接 PC) 时使用。</td>
</tr>
<tr>
<td>5V Power On LED</td>
<td>当开发板通电后 (USB 或外部 5V 供电)，该红色指示灯将亮起。</td>
</tr>
<tr>
<td>DC/DC 转换器</td>
<td>直流 5V 转 3.3V，输出电流最高可达 2A。</td>
</tr>
<tr>
<td>Board B 连接器</td>
<td>1 对双排插孔，用于连接 PoE 子板（B 板）。</td>
</tr>
<tr>
<td>IP101GRI (PHY)</td>
<td>物理层 (PHY) 单端口 10/100 快速以太网收发器 IP101GRI 芯片，允许开发人员实现与以太网线缆的物理层连接。PHY 与 ESP32 通过简化模块连接接口 (RMII) 实现连接。RMII 是 媒体独立接口 (MII) 的标准简化版本。PHY 可在 10/100Mbps 速率下支持 IEEE 802.3 / 802.3u 标准。</td>
</tr>
<tr>
<td>RJ45 端口</td>
<td>以太网数据传输端口。</td>
</tr>
<tr>
<td>网络变压器</td>
<td>网络变压器端口是位于以太网模块的一小部分，可保护设备，使其免受故障和电压瞬变影响，包括防止收发器芯片和线缆之间产生共模信号。同时它也可以在收发器与以太网设备之间提供电流隔离。</td>
</tr>
<tr>
<td>Link/Activity LED</td>
<td>2 个 LED (绿色和红色)，可分别显示 PHY 处于 “Link” 状态或 “Activity” 状态。</td>
</tr>
<tr>
<td>BOOT Button</td>
<td>下载按键，按 BOOt 键开，同时按一下 EN 键（此时不要松开 BOOt 键）进入 “固件下载” 模式，通过串口下载固件。</td>
</tr>
<tr>
<td>EN 按键</td>
<td>复位按键。</td>
</tr>
<tr>
<td>GPIO Header 1</td>
<td>由 6 个未引出通孔组成，可连接至 ESP32 的备用 GPIO。具体介绍，请见 GPIO Header 1。</td>
</tr>
</tbody>
</table>

备注：开发板支持固件自动下载模式。若遵循应用程序开发 章节中的步骤并使用了规定软件，则无需对 BOOT 或 EN 按键进行任何操作。

PoE 子板（B板） PoE 子板转换以太网电缆传输的电能 (PoE)，为以太网母板 (A板) 提供电源。PoE 子板（B板）的主要组件见 功能概述 中的功能框图。

PoE 子板（B板）具有以下特性：

- 支持 IEEE 802.3at 标准
- 电源输出：5V，1.4A

如需使用 PoE 功能，请用以太网电缆将以太网母板 (A板) 上的 RJ45 Port 连接至 PoE 的交换机。以太网母板 (A板) 检测到来自 PoE 子板 (B板) 的 5V 供电后，将从 USB 供电自动切换至 PoE 供电。

表2: 表格2 PoE 子板（B板）

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 板连接器</td>
<td>4 个排针（左侧）和排母（右侧），用于将 PoE 子板 (B板) 连接至 Ethernet board (A)。左侧的管脚接来自 PoE 交换机的电源，右侧的管脚即为以太网母板 (A板) 提供 5V 电源。</td>
</tr>
<tr>
<td>外部电源终端</td>
<td>PoE 子板 (B板) 可选电源 (2.6 ~ 54 V)。</td>
</tr>
</tbody>
</table>
设置选项 本节介绍用于 ESP32-Ethernet-Kit 开发板的硬件配置选项。

功能选择开关 拨码开关打开时，拨码开关将列出的 GPIO 路由到 FT2232H 以提供 JTAG 功能。拨码开关关闭时，GPIO 可以用于其他目的。

<table>
<thead>
<tr>
<th>拨码开关</th>
<th>GPIO 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO13</td>
</tr>
<tr>
<td>2</td>
<td>GPIO12</td>
</tr>
<tr>
<td>3</td>
<td>GPIO15</td>
</tr>
<tr>
<td>4</td>
<td>GPIO14</td>
</tr>
</tbody>
</table>

RMII 时钟源选择 RMII 工作模式下的以太网 MAC 和 PHY 需要一个公共的 50 MHz 同步时钟 (即 RMII 时钟)，它既可以由外部提供，也可以由内部的 ESP32 APLL 产生 (不推荐)。

备注：有关 RMII 时钟源选择的更多信息，请参见 ESP32-Ethernet-Kit V1.2 以太网母板（A 板）原理图，第 2 页的位置 D2。

PHY 侧提供 RMII 时钟 ESP32-Ethernet-Kit 默认配置为 IP101GRI 的 50M_CLKO 信号线提供 RMII 时钟。该时钟信号由 PHY 外侧连接的 25 MHz 无源晶振经过倍频产生。详情请参见下图。

请注意，系统上电时 RESET_N 旁的下拉电阻会将 PHY 置于复位状态。ESP32 需要通过 GPIO5 将 RESET_N 拉高才能启动 PHY。只有这样才能保证系统的正常上电，否则 ESP32 会存在一定几率进入下载模式 (当 REF_CLK_50M 时钟信号在 GPIO0 上电采样阶段刚好处于高电平)。

ESP32 APLL 内部提供的 RMII 时钟 另一种选择是从 ESP32 APLL 内部获取 RMII 时钟，请参见下图。来自 GPIO0 的时钟信号首先被反相，以解决传输线延迟的问题，然后提供给 PHY。

要实现此选项，用户需要在板子上移除或添加一些阻容元器件。有关详细信息，请参见 ESP32-Ethernet-Kit V1.2 以太网母板（A 板）原理图，第 2 页，位置 D2。请注意，如果 APLL 已经用于其他用途（如 I2S 外设），那么只能使用外部 RMII 时钟。

GPIO 分配 本节介绍了 ESP32-Ethernet-Kit 开发板特定接口或功能的 GPIO 分配情况。

IP101GRI (PHY) 接口 下表显示了 ESP32 (MAC) 与 IP101GRI (PHY) 的管脚对应关系。ESP32-Ethernet-Kit 的实现默认设置为简化媒体独立接口。
图 26: PHY 侧提供 RMII 时钟
图 27: ESP32 APLL 内部提供的 RMII 时钟
### Chapter 1. 快速入门

<table>
<thead>
<tr>
<th>No.</th>
<th>ESP32 脚号 (MAC)</th>
<th>IP101GRI (PHY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMII 接口</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>GPIO21</td>
<td>TX_EN</td>
</tr>
<tr>
<td>2</td>
<td>GPIO19</td>
<td>TXD[0]</td>
</tr>
<tr>
<td>3</td>
<td>GPIO22</td>
<td>TXD[1]</td>
</tr>
<tr>
<td>4</td>
<td>GPIO25</td>
<td>RXD[0]</td>
</tr>
<tr>
<td>5</td>
<td>GPIO26</td>
<td>RXD[1]</td>
</tr>
<tr>
<td>6</td>
<td>GPIO27</td>
<td>CRS_DV</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0</td>
<td>REF_CLK</td>
</tr>
<tr>
<td>串行管理接口</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>GPIO23</td>
<td>MDC</td>
</tr>
<tr>
<td>9</td>
<td>GPIO18</td>
<td>MDIO</td>
</tr>
<tr>
<td>PHY 复位</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>GPIO5</td>
<td>Reset_N</td>
</tr>
</tbody>
</table>

备注：ESP32 的 RMII 接口下的所有管脚分配都是固定的，不能通过 IOMUX 或 GPIO 矩阵进行更改。REF_CLK 仅可选择 GPIO0、GPIO16 或 GPIO17，且不可通过 GPIO 矩阵进行更改。

### GPIO Header 1 本连接器包括 ESP32-Ethernet-Kit 开发板上部分不用做他用的 GPIO。

<table>
<thead>
<tr>
<th>No.</th>
<th>ESP32 脚号</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO32</td>
</tr>
<tr>
<td>2</td>
<td>GPIO33</td>
</tr>
<tr>
<td>3</td>
<td>GPIO34</td>
</tr>
<tr>
<td>4</td>
<td>GPIO35</td>
</tr>
<tr>
<td>5</td>
<td>GPIO36</td>
</tr>
<tr>
<td>6</td>
<td>GPIO39</td>
</tr>
</tbody>
</table>

### GPIO Header 2 根据“说明”描述的不同情形，本连接器包含可用做他用的 GPIO。

<table>
<thead>
<tr>
<th>No.</th>
<th>ESP32 脚号</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO17</td>
<td>见下方说明 1</td>
</tr>
<tr>
<td>2</td>
<td>GPIO16</td>
<td>见下方说明 1</td>
</tr>
<tr>
<td>3</td>
<td>GPIO4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>GPIO2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GPIO13</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>6</td>
<td>GPIO12</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>7</td>
<td>GPIO15</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>8</td>
<td>GPIO14</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>9</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>10</td>
<td>3V3</td>
<td>3.3 V 电源</td>
</tr>
</tbody>
</table>

备注：
1. ESP32 芯片的 GPIO16 和 GPIO17 脚没有引出至 ESP32-WROVER-E 模组的管脚，因此无法使用。如需使用 ESP32 的 GPIO16 和 GPIO17 脚，建议更换其他不含 PSRAM 的模组，比如 ESP32-WROOM-32D 或 ESP32-SOLO-1。
2. 具体功能取决于功能选择开关的设置。
### GPIO 管脚分配总结

<table>
<thead>
<tr>
<th>ESP32-WROVER-E</th>
<th>IPT01GR1</th>
<th>UART</th>
<th>JTAG</th>
<th>GPIO</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_VP</td>
<td></td>
<td></td>
<td></td>
<td>IO36</td>
<td></td>
</tr>
<tr>
<td>S_VN</td>
<td></td>
<td></td>
<td></td>
<td>IO39</td>
<td></td>
</tr>
<tr>
<td>IO34</td>
<td></td>
<td></td>
<td></td>
<td>IO34</td>
<td></td>
</tr>
<tr>
<td>IO35</td>
<td></td>
<td></td>
<td></td>
<td>IO35</td>
<td></td>
</tr>
<tr>
<td>IO32</td>
<td></td>
<td></td>
<td></td>
<td>IO32</td>
<td></td>
</tr>
<tr>
<td>IO33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO25</td>
<td>RXD[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO26</td>
<td>RXD[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO27</td>
<td>CRS_DV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO14</td>
<td></td>
<td>TMS</td>
<td>IO14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO12</td>
<td></td>
<td>TDI</td>
<td>IO12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO13</td>
<td></td>
<td>TCK</td>
<td>IO13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO15</td>
<td></td>
<td>TDO</td>
<td>IO15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO2</td>
<td></td>
<td></td>
<td>IO2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO0</td>
<td>REF_CLK</td>
<td></td>
<td></td>
<td></td>
<td>See note 1</td>
</tr>
<tr>
<td>IO4</td>
<td></td>
<td></td>
<td>IO4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO16</td>
<td></td>
<td></td>
<td>IO16 (NC)</td>
<td></td>
<td>See note 2</td>
</tr>
<tr>
<td>IO17</td>
<td></td>
<td></td>
<td>IO17 (NC)</td>
<td></td>
<td>See note 2</td>
</tr>
<tr>
<td>IO5</td>
<td></td>
<td></td>
<td>Reset_N</td>
<td></td>
<td>See note 1</td>
</tr>
<tr>
<td>IO18</td>
<td></td>
<td></td>
<td>MDIO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO19</td>
<td></td>
<td></td>
<td>TXD[0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO21</td>
<td></td>
<td></td>
<td>TX_EN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RXD0</td>
<td></td>
<td></td>
<td>RXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXD0</td>
<td></td>
<td></td>
<td>TXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO22</td>
<td></td>
<td></td>
<td>TXD[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO23</td>
<td></td>
<td></td>
<td>MDC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 备注:

1. 为防止 ESP32 器 GPIO 的上电状态受 PHY 部分时钟输出的影响，PHY 部分的 RESITOR_B，默认为低，以关
   闭 PHY 部分时钟输出。上电后，用户可通过 GPIO5 控制 RESITOR_B 以打开该时钟输出。见 PHY 部分提
   供 RMII 时钟。对于无法通过 RESITOR_B 关闭时钟输出的 PHY，PHY 部分建议使用可在外置禁用/能使
   的有源晶振。分使用 RESITOR_B 类似，默认情况下晶振模块应禁用，并在上电后由 ESP32 开启。有关
   参考数据，请参见 ESP32-Ethernet-Kit V1.2 以太网母板（A 板）原理图。
2. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-E 模组的管脚，因此无法使
   用。若需使用 ESP32 的 GPIO16 和 GPIO17 管脚，建议更换其他不含 PSRAM 的模组，比如 ESP32-
   WROOM-32D 或 ESP32-SOLO-1。

### 应用程序开发

ESP32-Ethernet-Kit 上电前，请首先确认开发板完好无损。

### 初始设置

1. 首先，请以大母板（A 板）上的所有开关均拨至 ON 状态，使 功能选择开关 处于默认状态。
2. 为了方便应用程序的下载和调试，不要为开发板输入任何信号。
3. 此时可以连接 PoE 子板（B 板），但不要向 B 板连接任何外部电源。
4. 使用 USB 数据线将大母板（A 板）连接至 PC。
5. 将 电源开关 从 GND 拨至 5V0 一侧。此时，5V Power On LED 应点亮。

### 正式开始开发

现在，请前往 快速入门 中的 安装 章节，查看如何设置开发环境，并尝试将示例项目烧录
至您的开发板。

请务必在进入下一步前，确保您已完成上述所有步骤。

Submit Document Feedback
配置与加载以太网示例 在完成开发环境设置和开发板测试后，您可以配置并烧录`ethernet/basic`示例。本示例专门用于测试以太网功能，支持不同PHY，包括ESP32-Ethernet-Kit V1.2（点击放大）开发板使用的IP101GRI。

针对ESP32-Ethernet-Kit V1.1的主要修改

- 更正拨码开关周围GPIO编号丝印。
- C1、C2、C42 和 C43 更改 为20 pF。详细信息见 ESP32-Ethernet-Kit V1.2 以太网母板（A 板）原理图。
- 模组ESP32-WROVER-B替换为ESP32-WROVER-E。

ESP32-Ethernet-Kit 的其他版本

- ESP32-Ethernet-Kit V1.0 入门指南
- ESP32-Ethernet-Kit V1.1 入门指南

相关文档

- ESP32-Ethernet-Kit V1.2 以太网母板（A 板）原理图 (PDF)
- ESP32-Ethernet-Kit PoE 子板（B 板）原理图 (PDF)
- ESP32-Ethernet-Kit V1.2 以太网母板（A 板）PCB 布局图 (PDF)
- ESP32-Ethernet-Kit PoE 子板（B 板）PCB 布局图 (PDF)
- ESP32 技术规格书 (PDF)
- ESP32-WROVER-E 技术规格书 (PDF)
- JTAG 调试
- H/W 硬件参考

有关本开发板的更多设计文档，请联系我们的商务部门sales@espressif.com。

ESP32-Ethernet-Kit V1.0 入门指南

本指南介绍了如何使用ESP32-Ethernet-Kit 开发板以及配置相关功能。

ESP32-Ethernet-Kit 是一款以太网转 Wi-Fi 开发板，可为以太网设备赋予 Wi-Fi 连接功能。为了提供更灵活的电源选项，ESP32-Ethernet-Kit 也同时支持以太网供电 (PoE)。

准备工作

- ESP32-Ethernet-Kit V1.0 开发板
- USB 数据线（A 转 Micro-B）
- PC（Windows、Linux 或 macOS）

您可以跳过介绍部分，直接前往应用程序开发章节。

概述 ESP32-Ethernet-Kit 是一款来自乐鑫的开发板。

该开发板由以太网母板（A 板）和PoE 子板（B 板）两部分组成。其中以太网母板（A 板）贴蓝牙 / Wi-Fi 双模 ESP32-WROVER-B模组和单端口 10/100 快速以太网收发器 (PHY) IP101GRI。PoE 子板（B 板）提供以太网供电功能。ESP32-Ethernet-Kit 的 A 板可在不连接 B 板的情况下独立工作。

为了实现程序下载和监控，A 板还集成了一款先进多协议USB桥接器（FTDI FT2232H芯片），进而允许开发人员直接通过USB接口，使用JTAG对ESP32进行调试，无需额外的 JTAG适配器。

功能概述 ESP32-Ethernet-Kit 开发板的主要组件和连接方式见下。

功能说明 ESP32-Ethernet-Kit 开发板的主要组件、接口及控制方式见下。
Chapter 1

图 28: ESP32-Ethernet-Kit V1.0

图 29: ESP32-Ethernet-Kit 功能框图（点击放大）
图 30: ESP32-Ethernet-Kit - 以太网母板（A 板）布局（点击放大）

以太网母板（A 板） 下表将从图片右上角开始，以顺时针顺序介绍图中的主要组件。
## 章节 1. 快速入门

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-WROVER-B 模块</td>
<td>这款 ESP32 模块内置 64-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。</td>
</tr>
<tr>
<td>GPIO Header 2</td>
<td>由 5 个未引出通孔组成，可连接至 ESP32 的部分 GPIO。具体介绍，请见 GPIO Header 2。</td>
</tr>
<tr>
<td>流控</td>
<td>跳线帽，可接入开发板信号。具体介绍，请见 流控。</td>
</tr>
<tr>
<td>功能选择开关</td>
<td>DIP 开关，可配置 ESP32 部分 GPIO 的功能。具体介绍，请见 功能选择开关。</td>
</tr>
<tr>
<td>Tx/Rx LED</td>
<td>2 个 LED，可显示 UART 传输的状态。</td>
</tr>
<tr>
<td>GPIO Header 3</td>
<td>可连接至 ESP32 的部分 GPIO，根据 功能选择开关 的位置有不同功能。</td>
</tr>
<tr>
<td>FT2232H</td>
<td>FT2232H 多协议 USB 转串口桥接器。开发人员可通过 USB 接口对 FT2232H 芯片进行控制和编程，与 ESP32 建立连接。FT2232H 芯片可提供 USB-to-ITAG 接口功能，并在通道 B 提供 USB-to-Serial 接口功能，方便开发人员的应用开发与调试。见 ESP32-Ethernet-Kit V1.0 以太网母板（A 板）原理图。</td>
</tr>
<tr>
<td>USB 端口</td>
<td>USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。</td>
</tr>
<tr>
<td>电源开关</td>
<td>电源开关。拔向 Boot 按键一侧，开发板上电；拔离 Boot 按键一侧，开发板掉电。</td>
</tr>
<tr>
<td>5V Input</td>
<td>5V 电源接口建议仅在开发板自动运行 (未连接 PC) 时使用。仅用于全负荷工作下的后备电源。</td>
</tr>
<tr>
<td>5V Power On LED</td>
<td>当开发板通电后 (USB 或外部 5V 供电)，该红色指示灯将亮起。</td>
</tr>
<tr>
<td>DC/DC 转换器</td>
<td>直流 5V 转 3.3V，输出电流高达 2 A。</td>
</tr>
<tr>
<td>B 板连接器</td>
<td>1 对 2 针排针，用于连接 PoE 子板（B 板）。</td>
</tr>
<tr>
<td>IP101GRI (PHY)</td>
<td>物理层 (PHY) 单端口 10/100 快速以太网收发器 IP101GRI，允许开发人员实现与以太网线缆的物理层连接。PHY 与 ESP32 通过简化媒体独立接口 (RMII) 实现连接。RMII 是 媒体独立接口 (MI) 的简化版本。PHY 可在 10/100 Mbps 速率下支持 IEEE 802.3 / 802.3u 标准。</td>
</tr>
<tr>
<td>RJ45 端口</td>
<td>以太网数据传输端口。</td>
</tr>
<tr>
<td>网络变压器</td>
<td>网络变压器属于以太网物理层的一部分，可保护电路免受故障和电压瞬变影响，包括防止收发器芯片和线缆之间产生共模信号，同时它也可以在收发器与以太网设备之间提供电流隔离。</td>
</tr>
<tr>
<td>Link/Activity LED</td>
<td>4 个 LED（绿色和红色），分别显示 PHY 处于 “Link” 状态或 “Activity” 状态。</td>
</tr>
<tr>
<td>BOOT 按键</td>
<td>下载按键。按下 BOOT 键保持，同时按一下 EN 键（此时不要松开 BOOT 键）进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>CH_PU 按键</td>
<td>复位按键。</td>
</tr>
<tr>
<td>GPIO Header 1</td>
<td>由 6 个未引出通孔组成，可连接至 ESP32 的备用 GPIO。具体介绍，请见 GPIO Header 1。</td>
</tr>
</tbody>
</table>

**PoE 子板（B 板）** PoE 子板通过以太网电缆传输电能 (PoE)，为以太网母板（A 板）提供电源。PoE 子板（B 板）的主要组件见 功能概述 中的功能框图。

---

Espressif Systems 56  Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
PoE 子板（B 板）具有以下特性：

- 支持 IEEE 802.3at
- 电源输出：5 V，1.4 A

如需使用 PoE 功能，请用以太网线缆将以太网母板（A 板）上的 RJ45 端口连接至 PoE 的交换机。太网母板（A 板）检测到来自 PoE 子板（B 板）的 5 V 供电后，将从 USB 供电自动切换至 PoE 供电。

图 31: ESP32-Ethernet-Kit - PoE 子板（B 板）布局（点击放大）

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 板连接器</td>
<td>1 个 4 针排母，用于将 B 板连接至以太网母板（A 板）。</td>
</tr>
<tr>
<td>外部电源终端</td>
<td>PoE 子板（B 板）备用电源。</td>
</tr>
</tbody>
</table>

设置选项 本节介绍用于 ESP32-Ethernet-Kit 开发板的硬件配置选项。

功能选择开关 用户可使用功能选择开关配置特定 GPIO 管脚的功能。

<table>
<thead>
<tr>
<th>DIP SW</th>
<th>GPIO 管脚</th>
<th>管脚功能（DIP SW 开启状态）</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>GPIO14</td>
<td>连接至 FT2232H，提供 JTAG 功能</td>
</tr>
<tr>
<td>2.</td>
<td>GPIO12</td>
<td>连接至 FT2232H，提供 JTAG 功能</td>
</tr>
<tr>
<td>3.</td>
<td>GPIO13</td>
<td>连接至 FT2232H，提供 JTAG 功能</td>
</tr>
<tr>
<td>4.</td>
<td>GPIO15</td>
<td>连接至 FT2232H，提供 JTAG 功能</td>
</tr>
<tr>
<td>5.</td>
<td>GPIO4</td>
<td>连接至 FT2232H，提供 JTAG 功能</td>
</tr>
<tr>
<td>6.</td>
<td>GPIO2</td>
<td>连接至板上 25 MHz 晶振</td>
</tr>
<tr>
<td>7.</td>
<td>GPIO5</td>
<td>连接至 IP101GRI 的 RESET_N 输入</td>
</tr>
<tr>
<td>8.</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

用户还可以关闭 DIP DW，将部分 GPIO 用于其他用途。

流控 2 x 2 跳线连接器，可用于 UART 流控功能。
### GPIO 分配

本节介绍了 ESP32-Ethernet-Kit 开发板特定接口或功能的 GPIO 分配情况。

### IP101GRI (PHY) 接口

ESP32 (MAC) 与 IP101GRI (PHY) 的管脚对应关系见下。ESP32-Ethernet-Kit 开发板默认使用媒体独立接口 (RMII)。

<table>
<thead>
<tr>
<th>接口</th>
<th>ESP32 管脚 (MAC)</th>
<th>IP101GRI (PHY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMII</td>
<td>GPIO21</td>
<td>TX_EN</td>
</tr>
<tr>
<td></td>
<td>GPIO19</td>
<td>TXD[0]</td>
</tr>
<tr>
<td></td>
<td>GPIO22</td>
<td>TXD[1]</td>
</tr>
<tr>
<td></td>
<td>GPIO25</td>
<td>RXD[0]</td>
</tr>
<tr>
<td></td>
<td>GPIO26</td>
<td>RXD[1]</td>
</tr>
<tr>
<td></td>
<td>GPIO27</td>
<td>CRS_DV</td>
</tr>
<tr>
<td></td>
<td>GPIO20</td>
<td>REF_CLK</td>
</tr>
<tr>
<td>串联管理接口</td>
<td>GPIO23</td>
<td>MDC</td>
</tr>
<tr>
<td>9</td>
<td>GPIO18</td>
<td>MDIO</td>
</tr>
<tr>
<td>PHY 复位</td>
<td>GPIO5</td>
<td>Reset_N</td>
</tr>
</tbody>
</table>

**备注：** 除了 REF_CLK 之外, RMII 接口下的所有管脚分配都是固定的，不能通过 IOMUX 或 GPIO 矩阵进行更改。

### GPIO Header 1

本连接器包括 ESP32-Ethernet-Kit 开发板上部分不用做他用的 GPIO。
## ESP32 管脚

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>GPIO32</td>
</tr>
<tr>
<td>2.</td>
<td>GPIO33</td>
</tr>
<tr>
<td>3.</td>
<td>GPIO34</td>
</tr>
<tr>
<td>4.</td>
<td>GPIO35</td>
</tr>
<tr>
<td>5.</td>
<td>GPIO36</td>
</tr>
<tr>
<td>6.</td>
<td>GPIO39</td>
</tr>
</tbody>
</table>

### GPIO Header 2
本连接器包括具有特定 MII 功能的 GPIO（GPIO2 除外）。根据具体情况，部分以太网应用程序可能需要使用此功能。

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
<th>MII 功能</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>GPIO17</td>
<td>EMAC_CLK_180</td>
<td>见说明 1。</td>
</tr>
<tr>
<td>2.</td>
<td>GPIO16</td>
<td>EMAC_CLK_OUT</td>
<td>见说明 1。</td>
</tr>
<tr>
<td>3.</td>
<td>GPIO4</td>
<td>EMAC_TX_ER</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>GPIO2</td>
<td>n/a</td>
<td>见说明 2。</td>
</tr>
<tr>
<td>5.</td>
<td>GPIO5</td>
<td>EMAC_RX_CLK</td>
<td>见说明 2。</td>
</tr>
</tbody>
</table>

**备注：**
1. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GPIO16 和 GPIO17 管脚，建议更换其他不含 SPIRAM 的模组，比如 ESP32-WROOM-32D 或 ESP32-SOLO-1。
2. 具体功能取决于功能选择开关 的设置。

### GPIO Header 3
本连接器中 GPIO 的功能取决于功能选择开关 的设置。
### ESP32 管脚

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>GPIO15</td>
</tr>
<tr>
<td>2.</td>
<td>GPIO13</td>
</tr>
<tr>
<td>3.</td>
<td>GPIO12</td>
</tr>
<tr>
<td>4.</td>
<td>GPIO14</td>
</tr>
<tr>
<td>5.</td>
<td>GND</td>
</tr>
<tr>
<td>6.</td>
<td>3V3</td>
</tr>
</tbody>
</table>

### GPIO 管脚分配

<table>
<thead>
<tr>
<th>S_VP</th>
<th>IP101GRI</th>
<th>UART</th>
<th>JTAG</th>
<th>GPIO</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_VN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO34</td>
<td></td>
<td></td>
<td></td>
<td>IO34</td>
<td></td>
</tr>
<tr>
<td>IO35</td>
<td></td>
<td></td>
<td></td>
<td>IO35</td>
<td></td>
</tr>
<tr>
<td>IO32</td>
<td></td>
<td></td>
<td></td>
<td>IO32</td>
<td></td>
</tr>
<tr>
<td>IO33</td>
<td></td>
<td></td>
<td></td>
<td>IO33</td>
<td></td>
</tr>
<tr>
<td>IO25</td>
<td>RXD[0]</td>
<td></td>
<td></td>
<td>IO32</td>
<td></td>
</tr>
<tr>
<td>IO26</td>
<td>RXD[1]</td>
<td></td>
<td></td>
<td>IO33</td>
<td></td>
</tr>
<tr>
<td>IO27</td>
<td>CRS_DV</td>
<td></td>
<td></td>
<td>IO33</td>
<td></td>
</tr>
<tr>
<td>IO14</td>
<td></td>
<td>TMS</td>
<td></td>
<td>IO14</td>
<td></td>
</tr>
<tr>
<td>IO12</td>
<td></td>
<td>TDI</td>
<td></td>
<td>IO12</td>
<td></td>
</tr>
<tr>
<td>IO13</td>
<td></td>
<td>RTS</td>
<td>TCK</td>
<td>IO13</td>
<td></td>
</tr>
<tr>
<td>IO15</td>
<td></td>
<td>CTS</td>
<td>TDO</td>
<td>IO15</td>
<td></td>
</tr>
<tr>
<td>IO2</td>
<td></td>
<td></td>
<td></td>
<td>IO2</td>
<td>见下方说明 1 和说明 3。</td>
</tr>
<tr>
<td>IO0</td>
<td></td>
<td></td>
<td></td>
<td>IO2</td>
<td>见下方说明 2 和说明 3。</td>
</tr>
<tr>
<td>IO4</td>
<td></td>
<td></td>
<td>nTRST</td>
<td>IO4</td>
<td></td>
</tr>
<tr>
<td>IO16</td>
<td></td>
<td></td>
<td></td>
<td>IO16 (NC)</td>
<td>见下方说明 4。</td>
</tr>
<tr>
<td>IO17</td>
<td></td>
<td></td>
<td></td>
<td>IO17 (NC)</td>
<td>见下方说明 4。</td>
</tr>
<tr>
<td>IO5</td>
<td></td>
<td></td>
<td>Reset_N</td>
<td>IO5</td>
<td></td>
</tr>
<tr>
<td>IO18</td>
<td></td>
<td></td>
<td>MDIO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO19</td>
<td></td>
<td></td>
<td>TXD[0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO21</td>
<td></td>
<td></td>
<td>TX_EN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RXD0</td>
<td></td>
<td></td>
<td></td>
<td>RXD</td>
<td></td>
</tr>
<tr>
<td>TXD0</td>
<td></td>
<td></td>
<td></td>
<td>TXD</td>
<td></td>
</tr>
<tr>
<td>IO22</td>
<td></td>
<td></td>
<td>TXD[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO23</td>
<td></td>
<td></td>
<td></td>
<td>MDC</td>
<td></td>
</tr>
</tbody>
</table>

**备注：**

1. GPIO2 用于使能 PHY 的外部振荡器。
2. GPIO0 用于为 PHY 提供 50 MHz 基准时钟源。为了防止传输线路延迟对时钟相位带来的影响，该时钟信号将首先被反相，而后提供给 PHY。
3. 为防止 PHY 端 GPIO0 的上电状态受到时钟输出的影响，PHY 的外部晶振将在 ESP32 上电后通过 GPIO2 使能。
4. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GPIO16 和 GPIO17 管脚，建议更换其他不含 SPIRAM 的模组，比如 ESP32-WROOM-32D 或 ESP32-SOLO-1。

**应用程序开发** ESP32-Ethernet-Kit 上电前，请首先确认开发板完好无损。

**初始设置**

1. 首先，请将 以太网母板（A 板） 上的所有开关均拨至 ON 状态，即使 功能选择开关 处于默认状态。
2. 为了方便应用程序的下载和测试，此时请不要在开发板安装任何跳线帽，也不要为开发板接入任何信号。
3. 此时可以连接 PoE 子板（B 板），但不要向 B 板连接任何外部电源。
4. 使用 USB 数据线将 以太网母板（A 板） 连接至 PC。
5. 将 电源开关 从 GND 拨至 5V0 一侧，此时，5V Power On LED 应点亮。

**正式开始开发** 现在，请前往 快速入门 中的 安装 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

如需使用较早 GNU Make 编译系统，则请参考 安装 章节。

请务必在进入下一步前，确保您已完成上述所有步骤。

**配置与加载以太网示例** 在完成开发环境设置和开发板测试后，您可以配置并烧录 ethernet/basic 示例。本示例专门用于测试以太网功能，支持不同 PHY，包括 ESP32-Ethernet-Kit V1.0 开发板 使用的 IP101GRI。

**相关文档**

- ESP32-Ethernet-Kit V1.0 以太网母板（A 板）原理图 (PDF)
- ESP32-Ethernet-Kit V1.0 PoE 子板（B 板）原理图 (PDF)
- 《ESP32 技术规格书》 (PDF)
- 《ESP32-WROVER-B 技术规格书》 (PDF)
- JTAG 调试
- H/W 硬件参考

有关本开发板的更多设计文档，请联系我们的商务部门 sales@espressif.com。

**ESP32-Ethernet-Kit V1.1 入门指南**

本指南将介绍 ESP32-Ethernet-Kit 开发板的配置以及相关功能的使用。

ESP32-Ethernet-Kit 是一款以太网转 Wi-Fi 开发板。可为以太网设备赋予 Wi-Fi 连接功能。为了提供更灵活的电源选项，ESP32-Ethernet-Kit 同时也支持以太网供电 (PoE)。

**准备工作**

- ESP32-Ethernet-Kit V1.1 开发板
- USB 数据线 (A 转 Micro-B)
- PC (Windows, Linux 或 macOS)

您可以跳过介绍部分，直接前往 应用程序开发 章节。
Chapter 1. 快速入门

图 32: ESP32-Ethernet-Kit V1.1
概述 ESP32-Ethernet-Kit 是一款来自 乐鑫 的开发板。

它由 ESP32-WROVER-B 模组和单端口 10/100 Mbps 以太网收发器 (PHY) IP101GRI、PoE 子板（B 板）构成。ESP32-Ethernet-Kit 的 A 板可在不连接 B 板的情况下独立工作。

为了实现程序下载和监控，A 板还集成了一款先进多协议 USB 桥接器（FTDI FT2232H 芯片）。FTDI FT2232H芯片使得开发人员无需额外的 JTAG 适配器，通过 USB 桥接器使用 JTAG 接口便可对 ESP32 直接进行调试。

功能概述 ESP32-Ethernet-Kit 开发板的主要组件和连接方式如下。

![ESP32-Ethernet-Kit 功能框图](image)

图 33: ESP32-Ethernet-Kit 功能框图（点击放大）

功能说明 有关 ESP32-Ethernet-Kit 开发板的主要组件、接口及控制方式，请见下方的图片和表格。

以太网母板（A 板） 下表将从图片右上角开始，以顺时针顺序介绍图中的主要组件。
### 图 34: ESP32-Ethernet-Kit - 以太网母板 (A 板) 布局（点击放大）

### 表 3: 表格 1 组件介绍

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-WROVER-B 模组</td>
<td>这款 ESP32 模组内置 64-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。</td>
</tr>
<tr>
<td>GPIO Header 2</td>
<td>由 5 个未引出通孔组成，可连接至 ESP32 的部分 GPIO。具体介绍，请查阅 GPIO Header 2。</td>
</tr>
<tr>
<td>功能选择开关</td>
<td>一个 4 位拨码开关，可配置 ESP32 部分 GPIO 的功能。请注意，拨码开关旁边开发板的丝印层上的 GPIO 管脚标记的位置是不正确的。有关详细信息和正确的管脚分配，请见功能选择开关。</td>
</tr>
<tr>
<td>Tx/Rx LEDs</td>
<td>2 个 LED，可显示 UART 传输的状态。</td>
</tr>
<tr>
<td>FT2232H</td>
<td>FT2232H 多协议 USB 转串口桥接器，开发人员可通过 USB 接口对 FT2232H 芯片进行控制和编程，与 ESP32 建立连接。FT2232H 芯片可在通道 A 提供 USB-to-JTAC 接口功能，并在通道 B 提供 USB-to-Serial 接口功能，便利开发人员的应用开发与调试。见 ESP32-Ethernet-Kit V1.1 以太网母板 (A 板) 原理图。</td>
</tr>
<tr>
<td>USB 端口</td>
<td>USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。</td>
</tr>
<tr>
<td>电源开关</td>
<td>电源开关。拨向 5V0 按键侧，开发板上电；拨向 GND 按键一侧，开发板掉电。</td>
</tr>
<tr>
<td>5V Input</td>
<td>5 V 电源接口默认仅在开发板自动运行（未连接 PC）时使用。</td>
</tr>
<tr>
<td>5V Power On LED</td>
<td>当开发板通电后 (USB 或外部 5 V 供电)，该红色指示灯将亮起。</td>
</tr>
<tr>
<td>DC/DC 转换器</td>
<td>直流 5 V 转 3.3 V，输出电流最高可达 2 A。</td>
</tr>
<tr>
<td>Board B 连接器</td>
<td>1 号排针和排母，用于连接 PoE 子板（B 板）。</td>
</tr>
<tr>
<td>IP101GRI (PHY)</td>
<td>物理层 (PHY) 单端口 10/100 快速以太网收发器 IP101GRI 芯片，允许开发人员实现与以太网线缆的物理层连接。PHY 与 ESP32 通过简化媒体独立接口 (RMII) 实现连接。RMII 是 媒体独立接口 (MII) 的标准简化版本，PHY 可在 10/100 Mbps 速率下支持 IEEE 802.3 / 802.3u 标准。</td>
</tr>
<tr>
<td>RJ45 端口</td>
<td>以太网数据传输端口。</td>
</tr>
<tr>
<td>网络变压器</td>
<td>网络变压器属于以太网物理层的一部分，可保护电路，使其免受故障和电压瞬变影响。包括防止收发器芯片和线缆之间产生高共模信号。同时也可以在收发器与以太网设备之间提供电流隔离。</td>
</tr>
<tr>
<td>Link/Activity LED</td>
<td>2 个 LED（绿色和红色），可分别显示 PHY 处于“Link”状态或“Activity”状态。</td>
</tr>
<tr>
<td>BOOT Button</td>
<td>下载按键；按下 BOOT 键并保持，同时按一下 EN 键（此时不要松开 BOOT 键）进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>GPIO Header 1</td>
<td>由 6 个未引出通孔组成，可连接至 ESP32 的备用 GPIO。具体介绍，请查阅 GPIO Header 1。</td>
</tr>
</tbody>
</table>

Submit Document Feedback
PoE 子板（B 板） PoE 子板转换以太网电缆传输的电能 (PoE)，为以太网母板（A 板）提供电源。PoE 子板（B 板）的主要组件见功能概述 中的功能框图。

PoE 子板（B 板）具有以下特性：
- 支持 IEEE 802.3at 标准
- 电源输出：5 V，1.4 A

如需使用 PoE 功能，请用以太网线缆将以太网母板（A 板）上的 RJ45 Port 连接至 PoE 的交换机。以太网母板（A 板）检测到来自 PoE 子板（B 板）的 5 V 供电后，将从 USB 供电自动切换至 PoE 供电。

图 35: ESP32-Ethernet-Kit - PoE 子板（B 板）布局（点击放大）

### 表 4: 表格 2 PoE 子板（B 板）

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 板连接器</td>
<td>4 个排针（左侧）和排母（右侧），用于将 PoE 子板（B 板）连接至 Ethernet board (A)。左侧的管脚接受来自 PoE 交换机的电源。右侧的管脚为以太网母板（A 板）提供 5 V 电源。</td>
</tr>
<tr>
<td>外部电源终端</td>
<td>PoE 子板（B 板）可选电源 (26.6 ~ 54 V)。</td>
</tr>
</tbody>
</table>

### 设置选项
本节介绍用于 ESP32-Ethernet-Kit 开发板的硬件配置选项。

### 功能选择开关
拨码开关打开时，拨码开关将列出的 GPIO 路由到 FT2232H 以提供 JTAG 功能。拨码开关关闭时，GPIO 可用于其他目的。

<table>
<thead>
<tr>
<th>拨码开关</th>
<th>GPIO 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO13</td>
</tr>
<tr>
<td>2</td>
<td>GPIO12</td>
</tr>
<tr>
<td>3</td>
<td>GPIO15</td>
</tr>
<tr>
<td>4</td>
<td>GPIO14</td>
</tr>
</tbody>
</table>

备注：拨码开关旁边开发板的丝印层上的 GPIO 管脚标记的位置是不正确的。请以表格中的顺序为准。

### RMII 时钟源选择
RMII 工作模式下的以太网 MAC 和 PHY 需要一个公共的 50 MHz 同步时钟（即 RMII 时钟），它既可以由外部提供，也可以由内部的 ESP32 APPLL 产生。

备注：有关 RMII 时钟源选择的更多信息，请参见 ESP32-Ethernet-Kit V1.1 以太网母板（A 板）原理图，第 2 页的位置 D2。

### PHY 侧提供 RMII 时钟
ESP32-Ethernet-Kit 默认配置为 IP101GRI 的 50M_CLKO 信号线提供 RMII 时钟，该时钟信号由 PHY 外侧连接的 25MHz 无源晶振经过倍频产生。详情请参见下图。
图 36: PHY 侧提供 RMII 时钟
请注意，系统上电时 RESET_N 旁的下拉电阻会将 PHY 置于复位状态，ESP32 需要通过 GPIO5 将 RESET_N 拉高才能启动 PHY，只有这样才能保证系统的正常上电；否则 ESP32 会存在一定几率进入下载模式（当 REF_CLK_50M 时钟信号在 GPIO0 上电采样阶段刚好处于高电平）。

**ESP32 APLL 内部提供的 RMII 时钟** 另一种选择是从 ESP32 APLL 内部获取 RMII 时钟，请参见下图。来自 GPIO0 的时钟信号首先被反相，以解决传输线延迟的问题，然后提供给 PHY。

![ESP32 APLL 内部提供的 RMII 时钟](image)

图 37: ESP32 APLL 内部提供的 RMII 时钟

要实现此选项，用户需要在板子上移除或添加一些阻容元件件。有关详细信息，请参见 ESP32-Ethernet-Kit V1.1 以太网母板（A 板）原理图，第 2 页，位置 D2。请注意，如果 APLL 已经用于其他用途（如 I2S 外设），那么只能使用外部 RMII 时钟。

**GPIO 分配** 本节介绍了 ESP32-Ethernet-Kit 开发板特定接口或功能的 GPIO 分配情况。

**IP101GRI (PHY) 接口** 下表显示了 ESP32 (MAC) 与 IP101GRI (PHY) 的管脚对应关系。ESP32-Ethernet-Kit 的实现默认设置为简化媒体独立接口。
<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚 (MAC)</th>
<th>IP101GRI (PHY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO21</td>
<td>TX_EN</td>
</tr>
<tr>
<td>2</td>
<td>GPIO19</td>
<td>TXD[0]</td>
</tr>
<tr>
<td>3</td>
<td>GPIO22</td>
<td>TXD[1]</td>
</tr>
<tr>
<td>4</td>
<td>GPIO25</td>
<td>RXD[0]</td>
</tr>
<tr>
<td>5</td>
<td>GPIO26</td>
<td>RXD[1]</td>
</tr>
<tr>
<td>6</td>
<td>GPIO27</td>
<td>CRS_DV</td>
</tr>
<tr>
<td>7</td>
<td>GPIO0</td>
<td>REF_CLK</td>
</tr>
<tr>
<td>8</td>
<td>GPIO23</td>
<td>MDC</td>
</tr>
<tr>
<td>9</td>
<td>GPIO18</td>
<td>MDIO</td>
</tr>
<tr>
<td>10</td>
<td>GPIO5</td>
<td>Reset_N</td>
</tr>
</tbody>
</table>

备注：除了 REF_CLK 之外，ESP32 的 RMII 接口下的所有管脚分配都是固定的，不能通过 IOMUX 或 GPIO 阵列进行更改。

**GPIO Header 1**  本连接器包括 ESP32-Ethernet-Kit 开发板上部分不用做他用的 GPIO。

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO32</td>
</tr>
<tr>
<td>2</td>
<td>GPIO33</td>
</tr>
<tr>
<td>3</td>
<td>GPIO34</td>
</tr>
<tr>
<td>4</td>
<td>GPIO35</td>
</tr>
<tr>
<td>5</td>
<td>GPIO36</td>
</tr>
<tr>
<td>6</td>
<td>GPIO39</td>
</tr>
</tbody>
</table>

**GPIO Header 2**  根据“说明”描述的不同情形，本连接器包含可用做他用的 GPIO。

<table>
<thead>
<tr>
<th></th>
<th>ESP32 管脚</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO17</td>
<td>见下方说明 1</td>
</tr>
<tr>
<td>2</td>
<td>GPIO16</td>
<td>见下方说明 1</td>
</tr>
<tr>
<td>3</td>
<td>GPIO4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>GPIO2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GPIO13</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>6</td>
<td>GPIO12</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>7</td>
<td>GPIO15</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>8</td>
<td>GPIO14</td>
<td>见下方说明 2</td>
</tr>
<tr>
<td>9</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>10</td>
<td>3V3</td>
<td>3.3 V 电源</td>
</tr>
</tbody>
</table>

备注：
1. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GP1016 和 GPIO17 管脚，建议更换其他不含 PSRAM 的模组，例如 ESP32-WROOM-32D 或 ESP32-SOLO-1。
2. 具体功能取决于功能选择开关的设置。
### GPIO 管脚分配总结

<table>
<thead>
<tr>
<th></th>
<th>ESP32-WROVER-B</th>
<th>IPT01GRI</th>
<th>UART</th>
<th>JTAG</th>
<th>GPIO</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_VP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO36</td>
<td></td>
</tr>
<tr>
<td>S_VN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO39</td>
<td></td>
</tr>
<tr>
<td>IO34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO34</td>
<td></td>
</tr>
<tr>
<td>IO35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO35</td>
<td></td>
</tr>
<tr>
<td>IO32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO32</td>
<td></td>
</tr>
<tr>
<td>IO33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO33</td>
<td></td>
</tr>
<tr>
<td>IO25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RXD[0]</td>
<td></td>
</tr>
<tr>
<td>IO26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RXD[1]</td>
<td></td>
</tr>
<tr>
<td>IO27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CRS_DV</td>
<td></td>
</tr>
<tr>
<td>IO14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TMS</td>
<td>IO14</td>
</tr>
<tr>
<td>IO12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TDI</td>
<td>IO12</td>
</tr>
<tr>
<td>IO13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RTS</td>
<td>TCK IO13</td>
</tr>
<tr>
<td>IO15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CTS</td>
<td>TDO IO15</td>
</tr>
<tr>
<td>IO2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO2</td>
</tr>
<tr>
<td>IO0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REF_CLK</td>
<td>See note 1</td>
</tr>
<tr>
<td>IO4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO4</td>
</tr>
<tr>
<td>IO16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO16 (NC)</td>
<td>See note 2</td>
</tr>
<tr>
<td>IO17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO17 (NC)</td>
<td>See note 2</td>
</tr>
<tr>
<td>IO5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reset_N</td>
<td>See note 1</td>
</tr>
<tr>
<td>IO18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MDIO</td>
<td></td>
</tr>
<tr>
<td>IO19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TXD[0]</td>
<td></td>
</tr>
<tr>
<td>IO21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TX_EN</td>
<td></td>
</tr>
<tr>
<td>RXD0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RXD</td>
<td></td>
</tr>
<tr>
<td>TXD0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TXD</td>
<td></td>
</tr>
<tr>
<td>IO22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TXD[1]</td>
<td></td>
</tr>
<tr>
<td>IO23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MDC</td>
<td></td>
</tr>
</tbody>
</table>

### 备注:

1. 为防止 ESP32 的 GPIO 的上电状态受 PHY 随时钟输出的影响，PHY 侧 RESET_N 默认为低，以关闭 PHY 随时钟输出。上电后，您可通过 GPIO5 控制 RESET_N 以打开该时钟输出。请参见 phy 侧提供 RMI 时钟。对于无法通过 RESET_N 关闭时钟输出的 PHY，PHY 侧建议使用可在外部禁用/使能的有源晶振。与使用 RESET_N 类似，默认情况下晶振模块应禁用，并在上电后由 ESP32 开启。有关参考设计，请参见 ESP32-Ethernet-Kit V1.1 以太网母板（A 板）原理图。

2. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GPIO16 和 GPIO17 管脚，建议更换其他不含 PSRAM 的模组，如 ESP32-WROOM-32D 或 ESP32-SOLO-1。

### 应用程序开发

ESP32-Ethernet-Kit 上电前，请首先确认开发板完好无损。

### 初始设置

1. 首先，请将以太网母板（A 板）上的所有开关均拨至 ON 状态，使 功能选择开关 处于默认状态。
2. 为了方便应用程序的下载和测试，不要为开发板输入任何信号。
3. 此时可以连接 PoE 子板（B 板），但不要向 B 板连接任何外部电源。
4. 使用 USB 数据线将以太网母板（A 板）连接至 PC。
5. 将 电源开关 从 GND 拨至 5V0 一侧。此时，5V Power On LED 应点亮。

### 正式开始开发

现在，请前往 快速入门 中的 安装 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

*请务必在进入下一步前，确保您已完成上述所有步骤。*
配置与加载以太网示例  在完成开发环境设置和开发板测试后，您可以配置并烧录 ethernet/basic 示例。本示例专门用于测试以太网功能，支持不同 PHY，包括 ESP32-Ethernet-Kit V1.1 开发板使用的 IP101GRI。

针对 ESP32-Ethernet-Kit V1.0 的主要修改
- 原 GPIO0 反相后时钟供给 PHY 方案改为由 PHY 侧外接无源晶振，提供时钟给 GPIO0。原用于控制有源晶振的 OSC_EN 的 IO2 释放，可用作其他用途。
- 为防止 ESP32 侧 GPIO0 的上电状态受到 PHY 侧时钟输出的影响，PHY 侧 RESET_N 默认为低，关闭 PHY 侧时钟输出。而后再通过 GPIO0 控制 RESET_N 打开该时钟输出。
- 移除 FT2232H 芯片的外部 SPI Flash U6。
- 移除流控的测试排针 J4。
- 移除 nTRST JTAG 信号，相应的 GPIO4 可用作其他用途。
- GPIO15 线上的上拉电阻 R68 移至 JTAG 的 MTDO 侧。
- 为了加强 A 板和 B 板连接间的防呆设计（减少反向插入 B 板的机会），将原先 A 板上的 2 排 4 针排针改为 1 排 4 针排母和 1 排 4 针排针。相应的 4 针排针排和排母排则安装在 B 板上。

ESP32-Ethernet-Kit 的其他版本
- ESP32-Ethernet-Kit V1.0 入门指南

相关文档
- ESP32-Ethernet-Kit V1.1 以太网母板（A 板）原理图 (PDF)
- ESP32-Ethernet-Kit V1.0 PoE 子板（B 板）原理图 (PDF)
- ESP32 技术规格书 (PDF)
- ESP32-WROVER-B 技术规格书 (PDF)
- JTAG 调试
- H/W 硬件参考

有关本开发板的更多设计文档，请联系我们的商务部 sales@espressif.com。

ESP32-DevKitS(-R)

本指南介绍了乐鑫专为 ESP32 设计的烧录底板 ESP32-DevKitS(-R)。

ESP32-DevKitS(-R) 包括两款开发板，分别是 ESP32-DevKitS 和 ESP32-DevKitS-R。S 代表弹簧 (spring)，R 代表 WROVER 模组。

本指南包括如下内容：
- 入门指南: 简要介绍了 ESP32-DevKitS(-R) 和硬件、软件设置指南。
- 硬件参考: 详细介绍了 ESP32-DevKitS(-R) 的硬件。
- 相关文档: 列出了相关文档的链接。

Espressif Systems 70 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
**入门指南** 本节介绍了如何快速上手 ESP32-DevKitS(-R)。开始部分介绍了 ESP32-DevKitS(-R)，如何烧录开发板。小节介绍了怎样在 ESP32-DevKitS(-R) 上安装模块，配置及烧录固定。

**概述** ESP32-DevKitS(-R) 是乐鑫专为 ESP32 设计的烧录底板，无需将 ESP32 模组与电源和信号线焊接便可进行烧录。ESP32-DevKitS(-R) 搭载模块后，可作为类似 ESP32-DevKitC 的小型开发板使用。

ESP32-DevKitS 和 ESP32-DevKitS-R 仅在弹簧布局上有所不同，用于适配不同的 ESP32 模块。

- **ESP32-DevKitS:**
  - ESP32-WROOM-32
  - ESP32-WROOM-32D
  - ESP32-WROOM-32U
  - ESP32-SOLO-1
  - ESP32-WROOM-32E
  - ESP32-WROOM-32UE

- **ESP32-DevKitS-R:**
  - ESP32-WROVER (PCB & IPEX)
  - ESP32-WROVER-B (PCB & IPEX)
  - ESP32-WROVER-E
  - ESP32-WROVER-IE

有关上述模块的详细信息，请参考 ESP32 系列模块。

![ESP32-DevKitS.png](image-url)

图 38: ESP32-DevKitS - 正面

组件介绍
图 39: ESP32-DevKitS-R - 正面

<table>
<thead>
<tr>
<th>主要组件</th>
<th>介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>引脚弹簧</td>
<td>适配模组上的邮票孔，用于连接和固定模组。</td>
</tr>
<tr>
<td>2.54 mm 排母</td>
<td>连接板上搭载模组的引脚。关于排母的描述，请见针座小节。</td>
</tr>
<tr>
<td>USB-to-UART 桥接器</td>
<td>单片机 USB 转 UART 桥接器，可提供高达 3 Mbps 的传输速率。</td>
</tr>
<tr>
<td>LDO</td>
<td>5V-to-3.3V 低压差线型稳压器 (LDO)。</td>
</tr>
<tr>
<td>Micro-USB 连接器/Micro USB 端口</td>
<td>USB 接口。可用作开发板的供电电源或连接电脑和开发板的通信接口。</td>
</tr>
<tr>
<td>EN 钮</td>
<td>复位按钮。</td>
</tr>
<tr>
<td>Boot 钮</td>
<td>下载按钮。按下 Boot 钮的同时按一下 EN 钮进入“固件下载”模式，通过串口下载固件。</td>
</tr>
<tr>
<td>电源指示 LED</td>
<td>开发板通电后（USB 或供电电源），该指示灯亮起。</td>
</tr>
</tbody>
</table>

**如何烧录开发板** 通电前，请确保 ESP32-DevKitS(-R) 完好无损。

**必备硬件**

- 任选一款 ESP32 模组
- USB 2.0 数据线（标准 A 型转 Micro-B 型）
- 电脑（Windows、Linux 或 macOS）

**硬件设置** 请按照下列步骤将模组安装到 ESP32-DevKitS(-R) 烧录底板上：

- 将模组轻放到 ESP32-DevKitS(-R) 烧录底板上。确保模组上方的邮票孔与底板上的引脚弹簧对齐。
- 将模组向下压，听到“咔哒”一声即说明模组已经压入底板。
- 检查所有引脚弹簧是否均已卡进邮票孔中。如发现错位，可使用镊子将引脚弹簧拨入邮票孔中。
软件设置

首选方式 使用 ESP-IDF 开发框架是将二进制文件 (*.bin) 烧录到 ESP32-DevKitS(-R) 的首选方式。请前往快速入门，在安装一节查看如何快速设置开发环境，将应用程序烧录至 ESP32-DevKitS(-R)。

备选方式 Windows 用户也可以使用 Flash 下载工具 烧录二进制文件。请下载该工具，解压并参照 doc 文件夹内的使用指南。

备注:
1. 烧录二进制文件时需将 ESP32 芯片设置成固件下载模式。这一步骤可由烧录工具自动执行，也可按住 Boot 按键并点按 EN 按键手动执行。
2. 烧录完成后，烧录工具默认情况下会重启模组，运行已烧录的程序。

图 40: ESP32-DevKitS 尺寸 - 背面

底板尺寸

内含组件和包装

零售订单 如购买样品，每个 ESP32-DevKitS(-R) 底板将以防静电袋或零售商选择的其他方式包装。
零售订单请前往 https://www.espressif.com/zh-hans/company/contact/buy-a-sample。

批量订单 如批量购买，ESP32-DevKitS(-R) 烧录底板将以大纸板箱包装。
批量订单请前往 https://www.espressif.com/zh-hans/contact-us/sales-questions。

硬件参考
图 41: ESP32-DevKitS-R 尺寸 - 背面

功能框图 ESP32-DevKitS(-R) 的主要组件和连接方式如下图所示。

电源选项 您可从以下三种供电方式中任选其一给 ESP32-DevKitS(-R) 供电：
• Micro USB 端口供电（默认）
• 5V 和 GND 管脚供电
• 3V3 和 GND 管脚供电

建议选择第一种供电方式：Micro USB 端口供电。

<table>
<thead>
<tr>
<th>标签</th>
<th>信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>3V3</td>
</tr>
<tr>
<td>L2</td>
<td>EN</td>
</tr>
<tr>
<td>L3</td>
<td>VP</td>
</tr>
<tr>
<td>L4</td>
<td>VN</td>
</tr>
<tr>
<td>L5</td>
<td>34</td>
</tr>
<tr>
<td>L6</td>
<td>35</td>
</tr>
<tr>
<td>L7</td>
<td>32</td>
</tr>
<tr>
<td>L8</td>
<td>33</td>
</tr>
<tr>
<td>L9</td>
<td>25</td>
</tr>
<tr>
<td>L10</td>
<td>26</td>
</tr>
<tr>
<td>L11</td>
<td>27</td>
</tr>
<tr>
<td>L12</td>
<td>14</td>
</tr>
<tr>
<td>L13</td>
<td>12</td>
</tr>
<tr>
<td>L14</td>
<td>GND</td>
</tr>
<tr>
<td>L15</td>
<td>13</td>
</tr>
<tr>
<td>L16</td>
<td>D2</td>
</tr>
<tr>
<td>L17</td>
<td>D3</td>
</tr>
<tr>
<td>L18</td>
<td>CMD</td>
</tr>
<tr>
<td>L19</td>
<td>5V</td>
</tr>
</tbody>
</table>

下一页继续
表 5 - 续上页

<table>
<thead>
<tr>
<th>标签</th>
<th>信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>GND</td>
</tr>
<tr>
<td>R2</td>
<td>23</td>
</tr>
<tr>
<td>R3</td>
<td>22</td>
</tr>
<tr>
<td>R4</td>
<td>TX</td>
</tr>
<tr>
<td>R5</td>
<td>RX</td>
</tr>
<tr>
<td>R6</td>
<td>21</td>
</tr>
<tr>
<td>R7</td>
<td>GND</td>
</tr>
<tr>
<td>R8</td>
<td>19</td>
</tr>
<tr>
<td>R9</td>
<td>18</td>
</tr>
<tr>
<td>R10</td>
<td>5</td>
</tr>
<tr>
<td>R11</td>
<td>17</td>
</tr>
<tr>
<td>R12</td>
<td>16</td>
</tr>
<tr>
<td>R13</td>
<td>4</td>
</tr>
<tr>
<td>R14</td>
<td>0</td>
</tr>
<tr>
<td>R15</td>
<td>2</td>
</tr>
<tr>
<td>R16</td>
<td>15</td>
</tr>
<tr>
<td>R17</td>
<td>D1</td>
</tr>
<tr>
<td>R18</td>
<td>D0</td>
</tr>
<tr>
<td>R19</td>
<td>CLK</td>
</tr>
</tbody>
</table>

针座  针座图片请参照组件介绍。

相关文档

- ESP32-DevKitS(-R) 原理图 (PDF)
- ESP32 技术规格书 (PDF)
- ESP32-WROOM-32 技术规格书 (PDF)
- ESP32-WROOM-32D & ESP32-WROOM-32U 技术规格书 (PDF)
- ESP32-SOLO-1 技术规格书 (PDF)
- ESP32-WROVER 技术规格书 (PDF)
- ESP32-WROVER-B 技术规格书 (PDF)
- 乐鑫产品选型工具

ESP32-PICO-KIT-1

概述  ESP32-PICO-KIT-1 是一款基于 ESP32 的乐鑫开发板。

该开发板搭载 ESP32-PICO-V3（SiP，系统级封装）模组。ESP32-PICO-V3 具备完整的 Wi-Fi 和蓝牙功能，与其它 ESP32 系列模组相比，此模组已集成下述外围组件，无需额外安装：

- 40 MHz 晶振
- 4 MB flash
- 滤波电容
- 射频匹配网络

将上述组件封装至 ESP32-PICO-V3，用户无需额外购买，大大降低了安装和测试的成本，同时也增加了产品的可用性。

ESP32-PICO-KIT-1 开发板还集成了 USB-to-UART 桥接电路，方便开发人员直接通过 PC 的 USB 端口直接进行固件烧录和调试。

为方便硬件连接，ESP32-PICO-V3 上的所有 IO 信号和系统电源管脚均引出至开发板两侧焊盘 (18 x 0.1")。上述 18 个焊盘也均引出至排针，可使用杜邦线连接。
图 42: ESP32-DevKitS(-R)（点击放大）

备注：ESP32-PICO-KIT-1 开发板默认采用排针。

ESP32-PICO-KIT-1 为用户提供了基于 ESP32 芯片应用开发的硬件，更加方便用户探索 ESP32 芯片的功能。

本指南包括以下几个章节：

- 快速入门：简要介绍 ESP32-PICO-KIT-1 开发板，以及上手这款开发板所需的软件配置；
- 内含组件和包装方式：介绍零售订单和批量订单所提供的组件及包装方式；
- 硬件参考：提供 ESP32-PICO-KIT-1 开发板硬件详细信息；
- 硬件版本：介绍硬件历史版本（如有），已知问题（如有），并提供链接至历史版本开发板的入门指南（如有）；
- 相关文档：提供相关文档的链接。

快速入门 本节介绍了如何快速上手 ESP32-PICO-KIT-1。首先对 ESP32-PICO-KIT-1 开发板进行了简要介绍，然后在应用开发 小节介绍如何将固件烧录至开发板。

组件描述 ESP32-PICO-KIT-1 开发板的主要组件，接口及控制方式见下图。

从左上角开始，按顺时针顺序，开发板主要组件的描述见下表。
图 43: ESP32-PICO-KIT-1 外观图（点击放大）
图 44: ESP32-PICO-KIT-1 开发板布局 - 正面（点击放大）

<table>
<thead>
<tr>
<th>主要组件</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-PICO-V3</td>
<td>ESP32-PICO-KIT-1 开发板的板载模组 (SiP)，搭载 ESP32 SoC 芯片，仅需连接天线、LC 匹配电路、去耦电容和一个 EN 信号上拉电阻即可正常工作。</td>
</tr>
<tr>
<td>LDO</td>
<td>5 V 转 3.3 V 低电压稳压器</td>
</tr>
<tr>
<td>USB-to-UART</td>
<td>CP2102N：单芯片 USB-to-UART 桥接器，数据传输速率可达 3 Mbps。</td>
</tr>
<tr>
<td>Micro-B USB</td>
<td>USB 接口。可为开发板供电，或连接 PC 进行 USB 通信。</td>
</tr>
<tr>
<td>5V Power On LED</td>
<td>开发板上电后，红色指示灯亮起。更多信息，见相关文档中提供的原理图。</td>
</tr>
<tr>
<td>I/O Connector</td>
<td>ESP32-PICO-V3 上的管脚均已引出至开发板排针。用户可对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等功能。更多信息，见章节电脚描述。</td>
</tr>
<tr>
<td>BOOT 按键</td>
<td>下载按键。按下 Boot 按键并保持，同时按下 EN 键将进入固件下载模式，通过串口下载固件。</td>
</tr>
<tr>
<td>EN 按键</td>
<td>复位按键。</td>
</tr>
</tbody>
</table>

应用开发 上电前，请确保 ESP32-PICO-KIT-1 完好无损。

所需硬件
- 1 x ESP32-PICO-KIT-1 开发板
- 1 x USB 2.0 A 转 Micro B 数据线
- 1 x PC，运行 Windows、Linux 或 macOS 系统

软件配置 请前往快速入门，在安装一节查看如何快速配置开发环境，将应用程序烧录至 ESP32-PICO-KIT-1 开发板。

内含组件和包装方式

零售订单 如购买样品，每个 ESP32-PICO-KIT-1 将单独包装。
零售订单，请前往 https://www.espressif.com/en/contact-us/get-samples。
批量订单 如批量购买，ESP32-PICO-KIT-1 开发板将以大纸板箱包装。批量订单请前往 https://www.espressif.com/zh-hans/contact-us/sales-questions。

硬件参考

功能框图 ESP32-PICO-KIT-1 的主要组件和连接方式如下图所示。

图 45: ESP32-PICO-KIT-1 功能框图（点击放大）

供电选项 开发板提供以下三种供电方式，可任意选择一种：

- Micro USB 供电（默认）
- 5V/GND 管脚供电
- 3V3/GND 管脚供电

警告：上述供电方式只能选择一种方式，不能同时选择，否则可能会损坏开发板和/或电源。

管脚描述 组件描述 中所示的开发板两侧 I/O 管脚，其具体名称和功能见下表。管脚名称和编号与相关文档 中原理图保持一致。
## Chapter 1. 快速入门

### Header J2

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IO20</td>
<td>I/O</td>
<td>GPIO20</td>
</tr>
<tr>
<td>2</td>
<td>IO21</td>
<td>I/O</td>
<td>GPIO21, VSPIHD, EMAC_TX_EN</td>
</tr>
<tr>
<td>3</td>
<td>IO22</td>
<td>I/O</td>
<td>GPIO22, VSPIWP, U0RTS, EMAC_TXD1</td>
</tr>
<tr>
<td>4</td>
<td>IO19</td>
<td>I/O</td>
<td>GPIO19, VSPIQ, U0CTS, EMAC_TXD0</td>
</tr>
<tr>
<td>5</td>
<td>IO8</td>
<td>I/O</td>
<td>GPIO8, SD_DATA1, HS1_DATA1, U2CTS</td>
</tr>
<tr>
<td>6</td>
<td>IO7</td>
<td>I/O</td>
<td>GPIO7, SD_DATA0, HS1_DATA0, U2RTS</td>
</tr>
<tr>
<td>7</td>
<td>IO5</td>
<td>I/O</td>
<td>GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK</td>
</tr>
<tr>
<td>8</td>
<td>IO10</td>
<td>I/O</td>
<td>GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD</td>
</tr>
<tr>
<td>9</td>
<td>IO9</td>
<td>I/O</td>
<td>GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD</td>
</tr>
<tr>
<td>10</td>
<td>RXD0</td>
<td>I/O</td>
<td>GPIO3, U0RXD（注释1）, CLK_OUT2</td>
</tr>
<tr>
<td>11</td>
<td>TXD0</td>
<td>I/O</td>
<td>GPIO1, U0TXD（注释1）, CLK_OUT3, EMAC_RXD2</td>
</tr>
<tr>
<td>12</td>
<td>IO35</td>
<td>I</td>
<td>ADC1_CH7, RTC_GPIO5</td>
</tr>
<tr>
<td>13</td>
<td>IO34</td>
<td>I</td>
<td>ADC1_CH6, RTC_GPIO4</td>
</tr>
<tr>
<td>14</td>
<td>IO38</td>
<td>I</td>
<td>GPIO38, ADC1_CH2, RTC_GPIO2</td>
</tr>
<tr>
<td>15</td>
<td>IO37</td>
<td>I</td>
<td>GPIO37, ADC1_CH1, RTC_GPIO1</td>
</tr>
<tr>
<td>16</td>
<td>EN</td>
<td>I</td>
<td>CHIP_PU</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>P</td>
<td>接地</td>
</tr>
<tr>
<td>18</td>
<td>VDD33（3V3）</td>
<td>P</td>
<td>3.3V 供电</td>
</tr>
</tbody>
</table>

### Header J3

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>P</td>
<td>接地</td>
</tr>
<tr>
<td>2</td>
<td>SEN-SOR VP (FSVP)</td>
<td>I</td>
<td>GPIO36, ADC1_CH0, RTC_GPIO0</td>
</tr>
<tr>
<td>3</td>
<td>SEN-SOR VN (FSVN)</td>
<td>I</td>
<td>GPIO39, ADC1_CH3, RTC_GPIO3</td>
</tr>
<tr>
<td>4</td>
<td>IO25</td>
<td>I/O</td>
<td>GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0</td>
</tr>
<tr>
<td>5</td>
<td>IO26</td>
<td>I/O</td>
<td>GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1</td>
</tr>
<tr>
<td>6</td>
<td>IO32</td>
<td>I/O</td>
<td>32K_XP（注释2a）, ADC1_CH4, TOUCH9, RTC_GPIO9</td>
</tr>
<tr>
<td>7</td>
<td>IO33</td>
<td>I/O</td>
<td>32K_XN（注释2b）, ADC1_CH5, TOUCH8, RTC_GPIO8</td>
</tr>
<tr>
<td>8</td>
<td>IO27</td>
<td>I/O</td>
<td>GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV</td>
</tr>
<tr>
<td>9</td>
<td>IO14</td>
<td>I/O</td>
<td>ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2</td>
</tr>
<tr>
<td>10</td>
<td>IO12</td>
<td>I/O</td>
<td>ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI（注释3）, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3</td>
</tr>
<tr>
<td>11</td>
<td>IO13</td>
<td>I/O</td>
<td>ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER</td>
</tr>
<tr>
<td>12</td>
<td>IO15</td>
<td>I/O</td>
<td>ADC2_CH3, TOUCH3, RTC_GPIO13, MTD0, HSPICS0, HS2_CMD, SD_CMD, EMAC_RXD3</td>
</tr>
<tr>
<td>13</td>
<td>IO2</td>
<td>I/O</td>
<td>ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0</td>
</tr>
<tr>
<td>14</td>
<td>IO4</td>
<td>I/O</td>
<td>ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER</td>
</tr>
<tr>
<td>15</td>
<td>IO0</td>
<td>I/O</td>
<td>ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK</td>
</tr>
<tr>
<td>16</td>
<td>VDD33（3V3）</td>
<td>P</td>
<td>3.3V 供电</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>P</td>
<td>接地</td>
</tr>
<tr>
<td>18</td>
<td>EXT_5V（5V）</td>
<td>P</td>
<td>5V 供电</td>
</tr>
</tbody>
</table>
参考:
1. 该管脚已连接至板上 USB 桥接器芯片。
2. 32.768 kHz 晶振输入输出管脚：
   - (a) 输入管脚；
   - (b) 输出管脚。
3. ESP32-PICO-KIT-1 内置 SPI flash 工作电压为 3.3 V。因此，Strapping 管脚 MTDI 在上电复位过程中应拉低。如果连接了该管脚，请确保该管脚在复位前不要拉高。

图 46: ESP32-PICO-KIT-1 管脚布局（点击放大）

管脚布局

硬件版本 该开发板为最新硬件，尚未有历史版本。

相关文档
- ESP32-PICO-V3 技术规格书 (PDF)
- 乐鑫产品选型工具
- ESP32-PICO-KIT-1 原理图 (PDF)
- ESP32-PICO-KIT-1 PCB 布局图 (PDF)

有关本开发板的更多设计文档，请联系我们商务部门 sales@espressif.com。

ESP32-PICO-DevKitM-2

概述 ESP32-PICO-DevKitM-2 是一款基于 ESP32 的乐鑫开发板。板上搭载 ESP32-PICO-MINI-02(02U) 模组。ESP32-PICO-MINI-02(02U) 模组具备完整的 Wi-Fi 和蓝牙功能。

ESP32-PICO-DevKitM-2 开发板同时集成了 USB-to-UART 桥接电路，方便开发人员直接通过 PC 的 USB 端口直接进行固件烧录和调试。
为方便硬件连接，ESP32-PICO-MINI-02(02U)上的所有 IO 信号和系统电源管脚均引出至开发板两侧焊盘（18 x 0.1”）。上述 18 个焊盘也均引出至排针，可使用杜邦线连接。

备注：ESP32-PICO-DevKitM-2 开发板默认采用排针。

ESP32-PICO-DevKitM-2 为用户提供了基于 ESP32 芯片应用开发的硬件，更加方便用户探索 ESP32 芯片的功能。

图 47: ESP32-PICO-DevKitM-2 外观图（点击放大）

本指南包括以下几个章节：
- 快速入门：简介介绍 ESP32-PICO-DevKitM-2 开发板，以及上手这款开发板所需的软件配置；
- 内容组件和包装方式：介绍零售订单和批量订单所包含的组件及包装方式；
- 软件参考：提供 ESP32-PICO-DevKitM-2 开发板软件详细信息；
- 硬件组件：介绍硬件历史版本（如有）、已知问题（如有），并提供链接至历史版本开发板的入门指南（如有）。
- 相关文档：提供相关文档的链接。

快速入门 本节介绍了如何快速上手 ESP32-PICO-DevKitM-2。首先对 ESP32-PICO-DevKitM-2 开发板进行了简要介绍，然后在应用开发小节介绍如何将固件烧录至开发板。
图 48: ESP32-PICO-DevKitM-2 开发板组件布局 - 正面（点击放大）

主要组件 | 描述
--- | ---
ESP32-PICO-MINI-02 | ESP32-PICO-DevKitM-2 开发板的板载模组，搭载 ESP32 SoC 芯片。用户也可选择板载 ESP32-PICO-MINI-02U 的开发板。
LDO | 5V 转 3.3V 低压差稳压器
USB-to-UART | CP2102N：单芯片 USB-to-UART 桥接器，数据传输速率可达 3 Mbps。
Micro-B USB 接口 | USB 接口，可为开发板供电，或连接 PC 进行 USB 通信。
5V Power On LED | 开发板上电后，红色指示灯亮起。更多信息，见相关文档中提供的原理图。
I/O Connector | ESP32-PICO-MINI-02 上的管脚均引出至开发板排针。用户可对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等功能。更多信息，见章节管脚描述。
BOOT 按键 | 下载按键。按下 Boot 键并保持，同时按下 EN 键将进入固件下载模式，通过串口下载固件。
EN 按键 | 复位按键。

应用开发 上电前，请确保 ESP32-PICO-DevKitM-2 完好无损。

所需硬件
- 1 x ESP32-PICO-DevKitM-2 开发板
- 1 x USB 2.0 A 转 Micro B 数据线
- 1 x PC，运行 Windows、Linux 或 macOS 系统

软件配置 请前往快速入门，在安装一节查看如何快速配置开发环境，将应用程序烧录至 ESP32-PICO-DevKitM-2 开发板。

内含组件和包装方式

零售订单 如购买样品，每个 ESP32-PICO-DevKitM-2 将单独包装。
零售订单，请前往 https://www.espressif.com/en/contact-us/get-samples。
批量订单 如批量购买，ESP32-PICO-DevKitM-2 开发板将以大纸板箱包装。
批量订单请前往 https://www.espressif.com/zh-hans/contact-us/sales-questions。

硬件参考

功能框图 ESP32-PICO-DevKitM-2 的主要组件和连接方式如下图所示。

![ESP32-PICO-DevKitM-2 功能框图](image)

图 49: ESP32-PICO-DevKitM-2 功能框图（点击放大）

供电方式 开发板提供以下三种供电方式，可任意选择一种：
- Micro USB 供电（默认）
- 5V/GND 管脚供电
- 3V3/GND 管脚供电

警告: 上述供电方式只能选择一种方式，不能同时选择，否则可能会损坏开发板和/或电源。

管脚描述 组件描述 中所示的开发板两侧 I/O 管脚，其具体名称和功能见下表。管脚名称和编号与相关文档 中原理图保持一致。
### Header J2

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IO20</td>
<td>I/O</td>
<td>GPIO20</td>
</tr>
<tr>
<td>2</td>
<td>IO21</td>
<td>I/O</td>
<td>GPIO21, VSPIHD, EMAC_TX_EN</td>
</tr>
<tr>
<td>3</td>
<td>IO22</td>
<td>I/O</td>
<td>GPIO22, VSPIWP, U0RTS, EMAC_TXD1</td>
</tr>
<tr>
<td>4</td>
<td>IO19</td>
<td>I/O</td>
<td>GPIO19, VSPIQ, U0CTS, EMAC_TXD0</td>
</tr>
<tr>
<td>5</td>
<td>IO8</td>
<td>I/O</td>
<td>GPIO8, SD_DATA1, HS1_DATA1, U2CTS</td>
</tr>
<tr>
<td>6</td>
<td>IO7</td>
<td>I/O</td>
<td>GPIO7, SD_DATA0, HS1_DATA0, U2RTS</td>
</tr>
<tr>
<td>7</td>
<td>IO5</td>
<td>I/O</td>
<td>GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK</td>
</tr>
<tr>
<td>8</td>
<td>NC</td>
<td>-</td>
<td>NC</td>
</tr>
<tr>
<td>9</td>
<td>NC</td>
<td>-</td>
<td>NC</td>
</tr>
<tr>
<td>10</td>
<td>RXD0</td>
<td>I/O</td>
<td>GPIO3, U0RXD, (见注解1), CLK_OUT2</td>
</tr>
<tr>
<td>11</td>
<td>TXD0</td>
<td>I/O</td>
<td>GPIO1, U0TXD, (见注解1), CLK_OUT3, EMAC_RXD2</td>
</tr>
<tr>
<td>12</td>
<td>IO35</td>
<td>I</td>
<td>ADC1_CH7, RTC_GPIO5</td>
</tr>
<tr>
<td>13</td>
<td>IO34</td>
<td>I</td>
<td>ADC1_CH6, RTC_GPIO4</td>
</tr>
<tr>
<td>14</td>
<td>IO38</td>
<td>I</td>
<td>GPIO38, ADC1_CH2, RTC_GPIO2</td>
</tr>
<tr>
<td>15</td>
<td>IO37</td>
<td>I</td>
<td>GPIO37, ADC1_CH1, RTC_GPIO1</td>
</tr>
<tr>
<td>16</td>
<td>EN</td>
<td>I</td>
<td>CHIP_PU</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>P</td>
<td>接地</td>
</tr>
<tr>
<td>18</td>
<td>VDD33(3V3)</td>
<td>P</td>
<td>3.3 V 供电</td>
</tr>
</tbody>
</table>

### Header J3

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>P</td>
<td>接地</td>
</tr>
<tr>
<td>2</td>
<td>SEN-SOR_VP (FSVP)</td>
<td>I</td>
<td>GPIO36, ADC1_CH0, RTC_GPIO0</td>
</tr>
<tr>
<td>3</td>
<td>SEN-SOR_VN (FSVN)</td>
<td>I</td>
<td>GPIO39, ADC1_CH3, RTC_GPIO3</td>
</tr>
<tr>
<td>4</td>
<td>IO25</td>
<td>I/O</td>
<td>GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0</td>
</tr>
<tr>
<td>5</td>
<td>IO26</td>
<td>I/O</td>
<td>GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1</td>
</tr>
<tr>
<td>6</td>
<td>IO32</td>
<td>I/O</td>
<td>32K_XP (见注解2a), ADC1_CH4, TOUCH9, RTC_GPIO9</td>
</tr>
<tr>
<td>7</td>
<td>IO33</td>
<td>I/O</td>
<td>32K_XN (见注解2b), ADC1_CH5, TOUCH8, RTC_GPIO8</td>
</tr>
<tr>
<td>8</td>
<td>IO27</td>
<td>I/O</td>
<td>GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV</td>
</tr>
<tr>
<td>9</td>
<td>IO14</td>
<td>I/O</td>
<td>ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2</td>
</tr>
<tr>
<td>10</td>
<td>IO12</td>
<td>I/O</td>
<td>ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI (见注解3), HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3</td>
</tr>
<tr>
<td>11</td>
<td>IO13</td>
<td>I/O</td>
<td>ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER</td>
</tr>
<tr>
<td>12</td>
<td>IO15</td>
<td>I/O</td>
<td>ADC2_CH3, TOUCH3, RTC_GPIO13, MTD0, HSPICS0, HS2_CMD, SD_CMD, EMAC_RXD3</td>
</tr>
<tr>
<td>13</td>
<td>IO2</td>
<td>I/O</td>
<td>ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0</td>
</tr>
<tr>
<td>14</td>
<td>IO4</td>
<td>I/O</td>
<td>ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER</td>
</tr>
<tr>
<td>15</td>
<td>IO0</td>
<td>I/O</td>
<td>ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK</td>
</tr>
<tr>
<td>16</td>
<td>VDD33(3V3)</td>
<td>P</td>
<td>3.3V 供电</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>P</td>
<td>接地</td>
</tr>
<tr>
<td>18</td>
<td>EXT_5V(5V)</td>
<td>P</td>
<td>5V 供电</td>
</tr>
</tbody>
</table>
备注:

1. 该管脚已连接至板上 USB 伴侣芯片。
2. 32.768 kHz 振幅输入输出管脚:
   (a) 输入管脚;
   (b) 输出管脚;
3. ESP32-PICO-DevKitM-2 内置 SPI flash 的工作电压为 3.3 V。因此，Strapping 管脚 MTDI 在上电复位过程中应拉低。如果连接了该管脚，请确保该管脚在复位中不要拉高。

图 50: ESP32-PICO-DevKitM-2 管脚布局（点击放大）

硬件版本 该开发板为最新硬件，尚未有历史版本。

相关文档

- ESP32-PICO-MINI-02 & ESP32-PICO-MINI-02U 技术规格书 (PDF)
- 乐鑫产品选型工具
- ESP32-PICO-DevKitM-2 原理图 (PDF)
- ESP32-PICO-DevKitM-2 PCB 布局图 (PDF)

有关本开发板的更多设计文档，请联系我们的商务部门 sales@espressif.com。

ESP32-DevKitM-1

本指南将介绍如何使用 ESP32-DevKitM-1，并提供更多关于此开发板的信息。

ESP32-DevKitM-1 是乐鑫推出的一款基于 ESP32-MINI-I(1U) 模组的入门级开发板。板上模组大部分管脚
均已引出至两侧排针，用户可根据实际需求，通过跳线轻松连接多种外围设备，同时也可将开发板插在
面包板上使用。
Chapter 1. 快速入门

本指南包括：
- 快速入门：提供 ESP32-DevKitM-1 的简要概述及必要的硬件和软件信息。
- 硬件参考：提供 ESP32-DevKitM-1 的详细硬件信息。
- 相关文档：提供相关文档的链接。

快速入门 本节介绍如何开始使用 ESP32-DevKitM-1。主要包括三大部分：首先，介绍一些关于 ESP32-DevKitM-1 的基本信息，然后在应用程序开发 章节介绍如何进行硬件初始化，最后介绍如何为 ESP32-DevKitM-1 烧录固件。

概述 ESP32-DevKitM-1 开发板是一款小巧实用的开发板，具备以下特色功能：
- 集成了 ESP32-MINI-1 或 ESP32-MINI-1U 模组
- USB 转串口编程接口同时可为开发板供电
- 设有排针
- 设有重置按钮和固件下载模式激活按钮
- 以及其他组件

内含组件和包装

零售订单 如购买样品，每个 ESP32-DevKitM-1 开发板将以防静电袋或零售商选择的其他方式包装。
零售订单请前往 https://www.espressif.com/zh-hans/company/contact/buy-a-sample。

批量订单 如批量购买，ESP32-DevKitM-1 开发板将以大纸板箱包装。
批量订单请前往 https://www.espressif.com/zh-hans/contact-us/sales-questions。

组件介绍 ESP32-DevKitM-1 开发板的主要组件、接口及控制方式见下图。下文以板载 ESP32-MINI-1 的开发板为例进行说明。
图 51: ESP32-DevKitM-1 - 正面

<table>
<thead>
<tr>
<th>主要组件</th>
<th>基本介绍</th>
</tr>
</thead>
<tbody>
<tr>
<td>板载模块</td>
<td>ESP32-MINI-1 模组或 ESP32-MINI-1U 模组。ESP32-MINI-1 带有板载 PCB 天线；ESP32-MINI-1U 带有外部天线连接器。两款模组内的芯片均封装 4 MB flash，更多详情，请见《ESP32-MINI-1 &amp; ESP32-MINI-1U 技术规格书》</td>
</tr>
<tr>
<td>5 V 转 3.3 V LDO</td>
<td>电源转换器，输入 5 V，输出 3.3 V</td>
</tr>
<tr>
<td>Boot 键</td>
<td>下载按键。按下 Boot 键的同时按一下 Reset 键进入“固件下载”模式，通过串口下载固件</td>
</tr>
<tr>
<td>Reset 键</td>
<td>复位按键</td>
</tr>
<tr>
<td>Micro-USB 接口</td>
<td>USB 接口，可用作开发板的供电电源或 PC 和 ESP32 芯片的通信接口</td>
</tr>
<tr>
<td>USB 至 UART 桥接器</td>
<td>单芯片 USB 至 UART 桥接器，可提供高达 3 Mbps 的传输速率</td>
</tr>
<tr>
<td>3.3 V 电源指示灯</td>
<td>开发板连接 USB 电源后，该指示灯亮起。更多信息，请见相关文档中的原理图</td>
</tr>
<tr>
<td>I/O 连接器</td>
<td>所有可用 GPIO 端脚（除 Flash 的 SPI 总线）均已引出至开发板的排针。用户可对 ESP32 芯片编程，开发多种功能</td>
</tr>
</tbody>
</table>

**应用程序开发**

ESP32-DevKitM-1 上电前，请首先确认开发板完好无损。

**硬件准备**

- ESP32-DevKitM-1 开发板
- USB A / micro USB B 数据线
- PC（Windows，Linux 或 macOS）

**软件设置**

现在，请前往快速入门中的安装章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。
注意：对于 2021 年 12 月 2 日之前生产的 ESP32-DevKitM-1，其搭载的模组为单核 MCU 模组。请查看 PCN-2021-021 以确定开发板上搭载的模组是否为单核模组。对于搭载单核模组的开发板，烧录应用程序前，需要在 menuconfig 中使能单核模式（CONFIG_FREERTOS_UNICORE）。

硬件参考

图 52: ESP32-DevKitM-1

电源选项 开发板可从以下三种供电方式中任选其一：
- Micro USB 供电（默认）
- 5V / GND 管脚供电
- 3V3 / GND 管脚供电

警告：
- 上述供电模式 不可同时连接，否则可能会损坏开发板和/或电源。
- 建议选择第一种供电方式：Micro USB 端口供电。

管脚描述 开发板两侧 I/O 管脚，其具体名称和功能见下表。外设管脚分配请参考《ESP32 技术规格书》。

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>P</td>
<td>接地</td>
</tr>
<tr>
<td>2</td>
<td>3V3</td>
<td>P</td>
<td>3.3 V 电源</td>
</tr>
<tr>
<td>3</td>
<td>GPIO36, ADC1_CH0, RTC_GPIO0</td>
<td>I</td>
<td>GPIO36, ADC1_CH0, RTC_GPIO0</td>
</tr>
</tbody>
</table>
表6 - 续上页

<table>
<thead>
<tr>
<th>编号</th>
<th>名称</th>
<th>类型</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>I37</td>
<td>I</td>
<td>GPIO37, ADC1_CH1, RTC_GPIO1</td>
</tr>
<tr>
<td>5</td>
<td>I38</td>
<td>I</td>
<td>GPIO38, ADC1_CH2, RTC_GPIO2</td>
</tr>
<tr>
<td>6</td>
<td>I39</td>
<td>I</td>
<td>GPIO39, ADC1_CH3, RTC_GPIO3</td>
</tr>
<tr>
<td>7</td>
<td>RST</td>
<td>I</td>
<td>复位； 高电平： 使能； 低电平： 关闭</td>
</tr>
<tr>
<td>8</td>
<td>I34</td>
<td>I</td>
<td>GPIO34, ADC1_CH6, RTC_GPIO4</td>
</tr>
<tr>
<td>9</td>
<td>I35</td>
<td>I</td>
<td>GPIO35, ADC1_CH7, RTC_GPIO5</td>
</tr>
<tr>
<td>10</td>
<td>IO32</td>
<td>I/O</td>
<td>GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9</td>
</tr>
<tr>
<td>11</td>
<td>IO33</td>
<td>I/O</td>
<td>GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8</td>
</tr>
<tr>
<td>12</td>
<td>IO25</td>
<td>I/O</td>
<td>GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0</td>
</tr>
<tr>
<td>13</td>
<td>IO26</td>
<td>I/O</td>
<td>GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1</td>
</tr>
<tr>
<td>14</td>
<td>IO27</td>
<td>I/O</td>
<td>GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV</td>
</tr>
<tr>
<td>15</td>
<td>IO14</td>
<td>I/O</td>
<td>GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2</td>
</tr>
<tr>
<td>16</td>
<td>IO12</td>
<td>I/O</td>
<td>5V 电源</td>
</tr>
<tr>
<td>17</td>
<td>IO13</td>
<td>I/O</td>
<td>GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER</td>
</tr>
<tr>
<td>18</td>
<td>IO15</td>
<td>I/O</td>
<td>GPIO15, ADC2_CH3, TOUCH3, RTC_GPIO13, MTD0, HSPICS0, HS2_CMD, SD_CMD, EMAC_RXD3</td>
</tr>
<tr>
<td>19</td>
<td>IO0</td>
<td>I/O</td>
<td>GPIO0, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0</td>
</tr>
<tr>
<td>20</td>
<td>IO2</td>
<td>I/O</td>
<td>GPIO2, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK</td>
</tr>
<tr>
<td>21</td>
<td>IO4</td>
<td>I/O</td>
<td>GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER</td>
</tr>
<tr>
<td>22</td>
<td>IO9</td>
<td>I/O</td>
<td>GPIO9, HS1_DATA2, U1RXD, SD_DATA2</td>
</tr>
<tr>
<td>23</td>
<td>IO10</td>
<td>I/O</td>
<td>GPIO10, HS1_DATA3, U1TXD, SD_DATA3</td>
</tr>
<tr>
<td>24</td>
<td>IO5</td>
<td>I/O</td>
<td>GPIO5, HS1_DATA6, VSPICLK, HS1_DATA7, VSPICS0, EMAC_RX_CLK</td>
</tr>
<tr>
<td>25</td>
<td>IO18</td>
<td>I/O</td>
<td>GPIO18, HS1_DATA7, VSPICLK</td>
</tr>
<tr>
<td>26</td>
<td>IO23</td>
<td>I/O</td>
<td>GPIO23, HS1_STROBE, VSPID</td>
</tr>
<tr>
<td>27</td>
<td>IO19</td>
<td>I/O</td>
<td>GPIO19, VSPIQ, U0CTS, EMAC_TXD0</td>
</tr>
<tr>
<td>28</td>
<td>IO22</td>
<td>I/O</td>
<td>GPIO22, VSPICL, U0RTS, EMAC_TXD1</td>
</tr>
<tr>
<td>29</td>
<td>IO21</td>
<td>I/O</td>
<td>GPIO21, VSPILK, EMAC_TX_EN</td>
</tr>
<tr>
<td>30</td>
<td>TXD0</td>
<td>I/O</td>
<td>GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2</td>
</tr>
<tr>
<td>31</td>
<td>RXD0</td>
<td>I/O</td>
<td>GPIO3, U0RXD, CLK_OUT2</td>
</tr>
</tbody>
</table>

### 管脚布局

### 硬件修订历史
尚无版本升级历史。

### 相关文档
- 《ESP32-MINI-1 & ESP32-MINI-1U 技术规格书》 (PDF)
- ESP32-DevKitM-1 原理图 (PDF)
- ESP32-DevKitM-1 PCB 布局图 (PDF)
- ESP32-DevKitM-1 布局图 (DXF) - 可使用 Autodesk Viewer 查看

1 P: 电源； I: 输入； O: 输出。
2 MTD0, GPIO0, GPIO2, MTD1 和 GPIO5 为 Strapping 管脚。在芯片上电和系统复位过程中，Strapping 管脚根据管脚的二进制电压值控制芯片功能。Strapping 管脚的具体描述和应用，请参考《ESP32 技术规格书》 > 章节 Strapping 管脚。
图 53: ESP32-DevKitM-1（点击放大）

- 乐鑫产品选型工具
- 《ESP32 技术规格书》(PDF)

有关本开发板的更多设计文档，请联系我们的商务部门 sales@espressif.com。

### 1.2.2 软件

如需在 ESP32 上使用 ESP-IDF，请安装以下软件：

- 设置工具链，用于编译 ESP32 代码；
- 编译构建工具——CMake 和 Ninja 编译构建工具，用于编译 ESP32 应用程序；
- 获取 ESP-IDF 软件开发框架。该框架已经基本包含 ESP32 使用的 API（软件库和源代码）和运行工具链的脚本；
Chapter 1. 快速入门

1.3 安装

我们提供以下方法帮助安装所有需要的软件，可根据需要选择其中之一。

1.3.1 IDE

备注：建议您通过自己喜欢的集成开发环境（IDE）安装 ESP-IDF。

- Eclipse Plugin
- VSCode Extension

1.3.2 手动安装

请根据您的操作系统选择对应的手动安装流程。

Windows 平台工具链的标准设置

概述 ESP-IDF 需要安装一些必备工具，才能围绕 ESP32 构建固件，包括 Python、Git、交叉编译器、CMake 和 Ninja 编译工具等。

在本入门指南中，我们通过命令提示符进行有关操作。不过，您在安装 ESP-IDF 后还可以使用 Eclipse Plugin 或其他支持 CMake 的图形化工具 IDE。

备注：限定条件：- 请注意 ESP-IDF 和 ESP-IDF 工具的安装路径不能超过 90 个字符，安装路径过长可能导致构建失败。- Python 或 ESP-IDF 的安装路径中一定不能包含空格或括号。- 除非操作系统配置为支持 Unicode UTF-8，否则 Python 或 ESP-IDF 的安装路径中也不能包括特殊字符（非 ASCII 码字符）

系统管理员可以通过如下方式将操作系统配置为支持 Unicode UTF-8：控制面板-更改日期、时间或数字格式-管理选项卡-更改系统地域-勾选选项“Beta：使用 Unicode UTF-8 支持全球语言”-点击确定-重启电脑。

ESP-IDF 工具安装器 安装 ESP-IDF 必备工具最简易的方式是下载一个 ESP-IDF 工具安装器。
在线安装与离线安装的区别  在线安装程序非常小，可以安装 ESP-IDF 的所有版本。在安装过程中，安装程序只下载必要的依赖文件，包括 Git For Windows 安装器。在线安装程序会将下载的文件存储在缓存目录 `%userprofile%\espressif` 中。

离线安装程序不需要任何网络连接。安装程序中包含了所有需要的依赖文件，包括 Git For Windows 安装器。

安装内容  安装程序会安装以下组件：

- 内置的 Python
- 交叉编译器
- OpenOCD
- CMake 和 Ninja 编译工具
- ESP-IDF

安装程序允许将程序下载到现有的 ESP-IDF 目录。推荐将 ESP-IDF 下载到 `%userprofile%\Desktop\esp-idf` 目录下，其中 `%userprofile%` 代表家目录。

启动 ESP-IDF 环境  安装结束时，如果勾选了 Run ESP-IDF PowerShell Environment 或 Run ESP-IDF Command Prompt (cmd.exe)，安装程序会在选定的提示符窗口启动 ESP-IDF。

Run ESP-IDF PowerShell Environment:

![ESP-IDF PowerShell Environment](image)

图 54: 完成 ESP-IDF 工具安装向导时运行 Run ESP-IDF PowerShell Environment

Run ESP-IDF Command Prompt (cmd.exe):

使用命令提示符  在后续步骤中，我们将使用 Windows 的命令提示符进行操作。
图 55: ESP-IDF PowerShell

图 56: 完成 ESP-IDF 工具安装向导时运行 Run ESP-IDF Command Prompt (cmd.exe)
Chapter 1.

图 57: ESP-IDF 命令提示符窗口
Chapter 1. 快速入门

ESP-IDF 工具安装器可在“开始”菜单中，创建一个打开 ESP-IDF 命令提示符窗口的快捷方式。本快捷方式可以打开 Windows 命令提示符（即 cmd.exe），并运行 export.bat 脚本以设置各环境变量（如 PATH，IDF_PATH 等）。此外，您可还以通过 Windows 命令提示符使用各种已经安装的工具。

注意：本快捷方式仅适用于 ESP-IDF 工具安装器中指定的 ESP-IDF 路径。如果您的电脑上存在多个 ESP-IDF 路径（比如您需要不同版本的 ESP-IDF），您有以下两种解决方法：

1. 为 ESP-IDF 工具安装器创建的快捷方式创建一个副本，并将新快捷方式的 ESP-IDF 工作路径指定为您希望使用的 ESP-IDF 路径。
2. 或者，您可以运行 cmd.exe，并切换至您希望使用的 ESP-IDF 目录，然后运行 export.bat。注意，这种方法要求 PATH 中存在 Python 和 Git。如果您在使用时遇到有关“找不到 Python 或 Git” 的错误信息，请使用第一种方法。

### 开始使用 ESP-IDF
目前您已经具备了使用 ESP-IDF 的所有条件，接下来将介绍如何开始您的第一个工程。

本指南将帮助您完成使用 ESP-IDF 的第一步。按照本指南，您将使用 ESP32 创建第一个工程，并构建、烧录和监控设备输出。

**备注：** 如果您还未安装 ESP-IDF，请参照安装中的步骤，获取使用本指南所需的所有软件。

### 开始创建工程
现在，您可以准备开发 ESP32 应用程序了。您可以从 ESP-IDF 中 examples 目录下的 get-started/hello_world 工程开始。

**重要：** ESP-IDF 编译系统不支持 ESP-IDF 路径中带有空格。

将 get-started/hello_world 工程复制至您本地的 ~/esp 目录下：

```
cd %userprofile%\esp
xcopy /e /i %IDF_PATH%\examples\get-started\hello_world hello_world
```

**备注：** ESP-IDF 的 examples 目录下有一系列示例工程，您可以按照上述方法复制并运行其中的任何示例，也可以直接编译示例，无需进行复制。

### 连接设备
现在，请将您的 ESP32 开发板连接到 PC，并查看开发板使用的串口。

在 Windows 操作系统中，串口名称通常以 COM 开头。

有关如何查看串口名称的详细信息，请见与 ESP32 创建串口连接。

**备注：** 请记住串口名，您会在后续步骤中使用。

### 配置工程
请进入 hello_world 目录，设置 ESP32 为目标芯片，然后运行工程配置工具 menuconfig。

**Windows**

```
cd %userprofile%\esp\hello_world
idf.py set-target esp32
idf.py menuconfig
```
打开一个新工程后，应首先使用 `idf.py set-target esp32` 设置“目标”芯片。注意，此操作将
清除并初始化项目之前的编译和配置（如有）。您也可以直接将“目标”配置为环境变量（此时可跳过该步骤）。
更多信息，请见 `Select the Target Chip: set-target`。

正确操作上述步骤后，系统将显示以下菜单：

![工程配置菜单图](image)

图 58: 工程配置主窗口

您可以通过此菜单设置项目的具体变量，包括 Wi-Fi 网络名称、密码和处理器速度等。
`hello_world` 示例项目会以默认配置运行，因此在这一项目中，可以跳过使用 `menuconfig` 进行项目配置这一步骤。

<table>
<thead>
<tr>
<th>注意：如果您使用的是 ESP32-DevKitC（板载 ESP32-SOLO-1 模组）或 ESP32-DevKitM-1（板载 ESP32-MINI-1(1U) 模组），请在烧写示例程序前，前往 <code>menuconfig</code> 中使用单核模式 (<code>CONFIG_FREERTOS_UNICORE</code>)。</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>备注：您终端窗口中显示的菜单颜色可能会与上图不同。您可以通过选项 <code>--style</code> 来改变外观。请运行 <code>idf.py menuconfig --help</code> 命令，获取更多信息。</th>
</tr>
</thead>
</table>

如果您使用的是支持的开发板，可以通过板级支持包 (BSP) 来协助您的开发。更多信息，请见其他提示。

编译工程 请使用以下命令，编译烧录工程：

```bash
idf.py build
```

运行以上命令可以编译应用程序和所有 ESP-IDF 组件，接着生成引导加载程序、分区表和应用程序二进制文件。

```bash
$ idf.py build
Running cmake in directory /path/to/hello_world/build
Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
Warn about uninitialized values.
  -- Found Git: /usr/bin/git (found version "2.17.0")
  -- Building empty aws_iot component due to configuration
  -- Component names: ...
  -- Component paths: ...
```

(下页继续)
如果一切正常，编译完成后将生成.bin 文件。

**烧录到设备** 请运行以下命令，将刚刚生成的二进制文件烧录至您的 ESP32 开发板：

```
idf.py -p PORT flash
```

请将 PORT 替换为 ESP32 开发板的串口名称。如果 PORT 未经定义，`idf.py` 将尝试使用可用的串口自动连接。

更多有关 `idf.py` 参数的详情，请见 `idf.py`。

备注：勾选 `flash` 选项将自动编译并烧录工程，因此无需再运行 `idf.py build`。

若在烧录过程中遇到问题，请前往 **烧录故障排除** 或与 **ESP32 创建串口连接** 获取更多详细信息。

**常规操作** 在烧录过程中，您会看到类似如下的输出日志：

```
...  (more lines of build system output)

[527/527] Generating hello_world.bin
esptool.py v2.3.1
Project build complete. To flash, run this command:
.../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600 write_flash -
 --flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/hello_world.
 --build 0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/
 --partition-table.bin
or run 'idf.py -p PORT flash'
```

```
... esptool.py --chip esp32 -p /dev/ttyUSB0 -b 460800 --before=default_reset --
--after=hard_reset write_flash --flash_mode dio --flash_freq 40m --flash_size 2MB--
--0x8000 partition_table/partition-table.bin 0x1000 bootloader/bootloader.bin--
--0x10000 hello_world.bin
esptool.py v3.0-dev
Serial port /dev/ttyUSB0
Connecting...........
Chip is ESP32D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 40MHz
MAC: 24:0a:c4:05:b9:14
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds (effective 5962.8...
-kbit/s)... 
Hash of data verified.
Compressed 26096 bytes to 15408...
Writing at 0x00001000... (100 %)
Wrote 26096 bytes (15408 compressed) at 0x00001000 in 0.4 seconds (effective 546.7...
-kbit/s)... 
Hash of data verified.
Compressed 147104 bytes to 77364...
```
如果一切顺利，烧录完成后，开发板将会复位。应用程序“hello_world”开始运行。
如果您希望使用 Eclipse 或是 VS Code IDE，而非 idf.py，请参考 Eclipse Plugin，以及 VSCode Extension。

**监视输出** 您可以使用 `idf.py -p PORT monitor` 命令，监视“hello_world”工程的运行情况。注意，不要忘记将 PORT 替换为您的串口名称。

运行该命令后，IDF 监视器 应用程序将启动:

```
$ idf.py -p <PORT> monitor
Running idf_monitor in directory [...]/esp/hello_world/build
Executing "python [...]/esp-idf/tools/idf_monitor.py -b 115200 [...]/esp/hello_
-world/build/hello_world.elf"...
--- idf_monitor on <PORT> 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun 8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
est Jun 8 2016 00:22:57
...?
```

此时，您就可以在启动日志和诊断日志之后，看到打印的“Hello world!”了。

```
... Hello world!
  Restarting in 10 seconds...
  This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 1, 2MB...
  --- Minimum free heap size: 298968 bytes
  Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

您可使用快捷键 Ctrl+t]，退出 IDF 监视器。

如果 IDF 监视器在烧录后很快发生错误，或打印信息全是乱码（如下），很有可能是因您的开发板采用了 26 MHz 晶振，而 ESP-IDF，默认支持大多数开发板使用的 40 MHz 晶振。

```
%Xng@y\.!d(89yw)+Hh9n9~ /9@0L566P~Ek06e6ca650jA~zY011(0,1+6) e666(Xng@y!.DR+Zy(09jpih)@+z5Ymvp
```

此时，您可以:

1. 退出监视器。
2. 返回 menuconfig。
3. 进入 Component config -> Hardware Settings -> Main XTAL Config -> Main XTAL frequency 进行配置，将 `CONFIG_XTAL_FREQ_SEL` 设置为 26 MHz。
4. 重新编译和烧录应用程序。
在当前的 ESP-IDF 版本中，ESP32 支持的主晶振频率如下:

- 26 MHz
- 40 MHz

备注：您也可以运行以下命令，一次性执行构建、烧录和监视过程：

```
idf.py -p PORT flash monitor
```

此外，
- 请前往 IDF 监视器，了解更多使用 IDF 监视器的快捷键和其他详情。
- 请前往 idf.py，查看更多 idf.py 命令和选项。

恭喜，您已完成 ESP32 的入门学习！
现在，您可以尝试一些其他 examples，或者直接开发自己的应用程序。

重要：一些示例程序不支持 ESP32，因为 ESP32 中不包含所需的硬件。
在编译示例程序前请查看 README 文件中 Supported Targets 表格。如果表格中包含 ESP32，或者不存在这个表格，那么即表示 ESP32 支持这个示例程序。

其他提示

权限问题 /dev/ttyUSB0 使用某些 Linux 版本向 ESP32 烧录固件时，可能会出现 Failed to open port /dev/ttyUSB0 错误消息。此时可以将用户添加至 Linux Dialout 组。

兼容的 Python 版本 ESP-IDF 支持 Python 3.7 及以上版本，建议升级操作系统到最新版本从而更新 Python，也可选择从 sources 安装最新版 Python，或使用 Python 管理系统如 pyenv 对版本进行升级管理。

上手板级支持包 您可以使用 板级支持包 (BSP)，协助您在开发板上的原型开发。仅需要调用几个函数，便可以完成对特定开发板的初始化。
一般来说，BSP 支持开发板上所有硬件组件。除了管脚定义和初始化功能外，BSP 还附带如传感器、显示器、音频编码器等外部元件的驱动程序。
BSP 通过 IDF 组件管理器 发布，您可以前往 IDF 组件注册器 进行下载。
以下示例演示了如何将 ESP-WROVER-KIT BSP 添加到项目中:

```
idf.py add-dependency esp_wrover_kit
```

更多有关使用 BSP 的示例，请前往 BSP 示例文件夹。

擦除 flash ESP-IDF 支持擦除 flash。请运行以下命令，擦除整个 flash:

```
idf.py -p PORT erase-flash
```

若存在需要擦除的 OTA 数据，请运行以下命令:

```
idf.py -p PORT erase-otadata
```

擦除 flash 需要一段时间，在擦除过程中，请勿断开设备连接。
Chapter 1. 快速入门

相關文档：想要自定义安装流程的高阶用户可参照:

- 在 Windows 环境下更新 ESP-SDK 工具
- 与 ESP32 创建串口连接
- Eclipse Plugin
- VSCode Extension
- IDF 监视器

在 Windows 环境下更新 ESP-IDF 工具

使用脚本安装 ESP-IDF 工具：请从 Windows “命令提示符”窗口，切换至 ESP-IDF 的安装目录。然后运行:

```
install.bat
```

对于 Powershell，请切换至 ESP-IDF 的安装目录。然后运行:

```
install.ps1
```

该命令可下载并安装 ESP-IDF 所需的工具。如您已经安装了某个版本的工具，则该命令将无效。ESP-IDF 工具的下载安装位置由 ESP-IDF 工具安装器的设置决定，默认情况下为:

```
C:\Users\username\espressif
```

使用“导出脚本”将 ESP-IDF 工具添加至 PATH 环境变量：ESP-IDF 工具安装器将在“开始菜单”为“ESP-IDF 命令提示符”创建快捷方式。点击该快捷方式可打开 Windows 命令提示符窗口。您可在该窗口使用所有已安装的工具。

有些情况下，您正在使用的命令提示符窗口并不是通过快捷方式打开的，此时如果想要在该窗口使用 ESP-IDF，您可以根据下方步骤将 ESP-IDF 工具添加至 PATH 环境变量。

首先，请打开需要使用 ESP-IDF 的命令提示符窗口，切换至安装 ESP-IDF 的目录，然后执行 export.bat，具体命令如下:

```
cd %userprofile%\esp\esp-idf
export.bat
```

对于 Powershell 用户，请同样切换至安装 ESP-IDF 的目录，然后执行 export.ps1，具体命令如下:

```
cd ~/esp/esp-idf
export.ps1
```

运行完成后，您就可以通过命令提示符使用 ESP-IDF 工具了。

与 ESP32 创建串口连接

可以使用 USB 至 UART 桥，与 ESP32 创建串口连接。

部分开发板中已经安装有 USB 至 UART 桥。如未安装，可使用外部桥。

安装有 USB 至 UART 桥的开发板：在安装有 USB 至 UART 桥的开发板中，PC 和桥之间通过 USB 连接，桥和 ESP32 之间通过 UART 连接。

外部 USB 至 UART 桥：部分开发板使用外部 USB 至 UART 桥。这种情况通常出现在需要控制空间和成本的产品中，例如一些小型开发板或成品。

使用 UART 进行烧录：本节描述如何使用 USB 至 UART 桥在 ESP32 和 PC 之间建立串行连接。板上桥与外部桥均适用。
图 59: 安装有 USB 至 UART 桥的开发板

图 60: 外部 USB 至 UART 桥

连接 ESP32 和 PC 用 USB 线将 ESP32 开发板连接到 PC。如果设备驱动程序没有自动安装，请先确认 ESP32 开发板上的 USB 至 UART 桥 (或外部转 UART 适配器) 型号，然后在网上搜索驱动程序，并进行手动安装。

以下是乐鑫 ESP32 开发板驱动程序的链接:

- CP210x: CP210x USB 至 UART 桥 VCP 驱动程序
- FTDI: FTDI 虚拟 COM 端口驱动程序

以上驱动仅供参考，请查看开发板用户指南，了解开发板具体使用的 USB 至 UART 桥芯片。一般情况下，当 ESP32 开发板与 PC 连接时，对应驱动程序应该已经被打包在操作系统中，并已经自动安装。

对于使用 USB 至 UART 桥下载的设备，您可以运行以下命令，包括定义波特率的可选参数。

```
idf.py -p PORT [-b BAUD] flash
```

如需改变烧录器的波特率，请用需要的波特率代替 BAUD。默认的波特率为 460800。

备注：如果设备不支持自动下载模式，则需要手动进入下载模式。请按住 BOOT 按钮，同时按一下 RESET 按钮。之后，松开 BOOT 按钮。

在 Windows 上查看端口 检查 Windows 设备管理器中的 COM 端口列表，断开 ESP32 与 PC 的连接，然后重新连接，查看哪个端口从列表中消失后又再次出现。

以下为 ESP32 DevKitC 和 ESP32 WROVER KIT 串口:
图 61: 设备管理器中 ESP32-DevKitC 的 USB 至 UART 桥
图 62: Windows 设备管理器中 ESP-WROVER-KIT 的两个 USB 串行端口
在 Linux 和 macOS 上查看端口  查看 ESP32 开发板（或外部转串口适配器）的串口设备名称，请将以下命令运行两次。首先，断开开发板或适配器，首次运行以下命令；然后，连接开发板或适配器，再次运行以下命令。其中，第二次运行命令后出现的端口即是 ESP32 对应的串口。

Linux:
```bash
ls /dev/tty*
```

macOS:
```bash
ls /dev/cu.*
```

备注：对于 macOS 用户：若没有看到串口，请检查是否安装 USB/串口驱动程序。具体应使用的驱动程序，见章节连接 ESP32 和 PC。对于 macOS High Sierra (10.13) 的用户，你可能还需要手动允许驱动程序的加载，具体可打开系统偏好设置 -> 安全和隐私 -> 通用，检查是否有信息显示：“来自开发人员的系统软件…”，其中开发人员的名称为 Silicon Labs 或 FTDI。

在 Linux 中添加用户到 dialout  当前登录用户应当可以通过 USB 对串口进行读写操作。在多数 Linux 版本中，您都可以通过以下命令，将用户添加到 dialout 组，从而获许读写权限:

```bash
sudo usermod -a -G dialout $USER
```

在 Arch Linux 中，需要通过以下命令将用户添加到 uucp 组中:
```bash
sudo usermod -a -G uucp $USER
```

请重新登录，确保串口读写权限生效。

确认串口连接  现在，请使用串口终端程序，查看重置 ESP32 后终端上是否有输出，从而验证串口连接是否可用。

ESP32 的控制台波特率默认为 115200。

Windows 和 Linux 操作系统  在本示例中，我们将使用 PuTTY SSH Client，PuTTY SSH Client 既可用于 Windows 也可用于 Linux。您也可以使用其他串口程序并设置如下的通信参数。

运行终端，配置在上述步骤中确认的串口：波特率 = 115200（如有需要，请更改为使用芯片的默认波特率），数据位 = 8，停止位 = 1，奇偶校验 = N。以下截屏分别展示了如何在 Windows 和 Linux 中配置串口和上述通信参数（如 115200-8-1-N）。注意，这里一定要选择在上述步骤中确认的串口进行配置。

然后，请检查 ESP32 是否有打印日志。如有，请在终端打开串口进行查看。这里的日志内容取决于加载到 ESP32 的应用程序，请参考输出示例。

备注：请在验证串口通信正常后，关闭串口终端。如果您让终端一直保持打开的状态，之后上传固件时将无法访问串口。

macOS 操作系统  macOS 提供了屏幕命令，因此您不用安装串口终端程序。

- 参考在 Linux 和 macOS 上查看端口，运行以下命令:
  ```bash
  ls /dev/cu.*
  ```

- 您会看到类似如下输出:
  ```bash
  /dev/cu.Bluetooth-Incoming-Port /dev/cu.SLAB_USBtoUART /dev/cu.SLAB_USBtoUART7
  ```
图 63: 在 Windows 操作系统中使用 PuTTY 设置串口通信参数
图 64: 在 Linux 操作系统中使用 PuTTY 设置串口通信参数
Chapter 1. 快速入门

- 根据您连接到电脑上的开发板类型和数量，输出结果会有所不同。请选择开发板的设备名称，并运行以下命令（如有需要，请将 “115200” 更改为使用芯片的默认波特率）：

```
screen /dev/cu.device_name 115200
```

将 device_name 替换为运行 ls /dev/cu.* 后出现的设备串口号。

- 您需要的正是 屏幕 显示的日志。日志内容取决于加载到 ESP32 的应用程序，请参考 输出示例。请使用 Ctrl-A + \ 键退出 屏幕 会话。

备注：请在验证完串口通信正常后，关闭 屏幕 会话。如果直接关闭终端窗口而没有关闭 屏幕，之后上传固件时将无法访问串口。

输出示例 以下是一个日志示例。如果没看到任何输出，请尝试重启开发板。

```
ets Jun 8 2016 00:22:57
rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
```

```
ets Jun 8 2016 00:22:57
rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
```

```
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0008,len:8
load:0x3fff0010,len:3464
load:0x40078000,len:7828
load:0x40080000,len:252
entry 0x40080034
I (44) boot: ESP-IDF v2.0-rc1-401-gf9fba35 2nd stage bootloader
I (45) boot: compile time 18:48:10
...
```

如果打印出的日志是可读的（而不是乱码），则表示串口连接正常。此时，您可以继续进行安装，并最终将应用程序上载到 ESP32。

备注：在某些串口接线方式下，在 ESP32 启动并开始打印串口日志前，需要在终端程序中禁用串口 RTS & DTR 管脚。该问题仅存在于将 RTS & DTR 管脚直接连接到 EN & GPIO0 管脚上的情况，绝大多数开发板（包括乐鑫所有的开发板）都没有这个问题。更多详细信息，请参考 esptool 文档。

如您在安装 ESP32 硬件开发的软件环境时，从 第五步：开始使用 ESP-IDF 吧 跳转到了这里，请从 第五步：开始使用 ESP-IDF 吧 继续阅读。

烧录故障排除

连接失败 如果在运行给定命令时出现如 “连接失败” 这样的错误，造成该错误的原因之一可能是运行 esptool.py 时出现错误。esptool.py 是构建系统调用的程序，用于重置芯片、与 ROM 引导加载器交互以及烧录固件的工具。可以按照以下步骤进行手动复位，轻松解决该问题。如果问题仍未解决，请参考 esptool 故障排除 获取更多信息。

esptool.py 通过使 USB 至 UART 桥（如 FTDI 或 CP210x）的 DTR 和 RTS 控制线生效来自动复位 ESP32（请参考与 ESP32 创建串口连接 获取更多详细信息）。DTR 和 RTS 控制线又连接到 ESP32 的 GPIO0 和 CHIP_FU (EN) 管脚上，因此 DTR 和 RTS 的电压电平变化会使 ESP32 进入固件下载模式。相关示例可查看 ESP32 DevKitC 开发板的 原理图。

一般来说，使用官方的 ESP-IDF 开发板不会出现问题。但是，esptool.py 在以下情况下不能自动重置硬件：

Espressif Systems 108 Release v5.1-dev-2066-g7869f4e151
Chapter 1. 快速入门

- 您的硬件没有连接到 GPIO0 和 CIHP_PU 的 DTR 和 RTS 控制线。
- DTR 和 RTS 控制线的配置方式不同。
- 不存在这样的串行控制线路。

根据硬件的种类，也可以将您的 ESP32 开发板手动设置为固件下载模式 (复位)。

- 对于乐鑫开发板，您可以参考对应开发板的入门指南或用户指南，例如，可以通过按住 Boot 按钮 (GPIO0) 再按住 EN 按钮 (CHIP_PU) 来手动复位 ESP-IDF 开发板。
- 对于其他类型的硬件，可以尝试将 GPIO0 拉低。

IDF 监视器

IDF 监视器是一个串行终端程序，用于收发目标设备串口的串行数据，IDF 监视器同时还兼具 IDF 的其他特性。

在 IDF 中调用 idf.py monitor 可以启用此监视器。

操作快捷键 为了方便与 IDF 监视器进行交互，请使用表中给出的快捷键。
<table>
<thead>
<tr>
<th>快捷键</th>
<th>操作</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+J</td>
<td>退出监视器程序</td>
<td></td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>菜单退出键</td>
<td>按下如下给出的任意键之一，并按指示操作。</td>
</tr>
<tr>
<td>• Ctrl+T</td>
<td>将菜单字符发送至远程</td>
<td></td>
</tr>
<tr>
<td>• Ctrl+]</td>
<td>将 exit 字符发送至远程</td>
<td></td>
</tr>
<tr>
<td>• Ctrl+P</td>
<td>重置目标设备，进入引导加载程序，通过 RTS 线暂停应用程序</td>
<td>重置目标设备，通过 RTS 线（如已连接）进入引导加载程序，此时开发板不运行任何程序。等待其他设备启动时可以使用此操作。</td>
</tr>
<tr>
<td>• Ctrl+R</td>
<td>通过 RTS 线重置目标设备</td>
<td>重置设备，并通过 RTS 线（如已连接）重新启动应用程序。</td>
</tr>
<tr>
<td>• Ctrl+F</td>
<td>编译并烧录此项目</td>
<td>暂停 idf_monitor，运行 flash 目标，然后恢复 idf_monitor。任何改动的源文件都会被重新编译，然后重新烧录。如果 idf_monitor 是以参数 -E 启动的，则会运行目标 encrypted-flash。</td>
</tr>
<tr>
<td>• Ctrl+A （或 A）</td>
<td>仅编译及烧录应用程序</td>
<td>暂停 idf_monitor，运行 app-flash 目标，然后恢复 idf_monitor。这与 flash 类似，但只有主应用程序被编译并被重新烧录。如果 idf_monitor 是以参数 -E 启动的，则会运行目标 encrypted-flash。</td>
</tr>
<tr>
<td>• Ctrl+Y</td>
<td>停止/恢复在屏幕上打印日志输出</td>
<td>激活时，会丢弃所有传入的串行数据。允许在不退出监视器的情况下快速暂停和检查日志输出。</td>
</tr>
<tr>
<td>• Ctrl+L</td>
<td>停止/恢复向文件写入日志输出</td>
<td>在工程目录下创建一个文件，用于写入日志输出。可使用快捷键停止/恢复该功能（退出 IDF 监视器也会终止该功能）。</td>
</tr>
<tr>
<td>• Ctrl+I （或者 I）</td>
<td>停止/恢复打印时间标记</td>
<td>IDF 监视器可以在每一行的开头打印一个时间标记。时间标记的格式可以通过 --timestamp-format 命令行参数来改变。</td>
</tr>
<tr>
<td>• Ctrl+H （或者 H）</td>
<td>显示所有快捷键</td>
<td></td>
</tr>
<tr>
<td>• Ctrl+X （或者 X）</td>
<td>退出监视器程序</td>
<td></td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>中断正在运行的应用程序</td>
<td>暂停 IDF 监视器并运行 GDB 项目调试器。这需要开启 ref:CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME 选项。</td>
</tr>
</tbody>
</table>

除了 Ctrl-] 和 Ctrl-T，其他快捷键信号会通过串口发送到目标设备。

### 兼具 IDF 特性

**自动解码地址** ESP-IDF 输出形式为 `0x4_________` 的十六进制代码地址后，IDF 监视器将使用 `addr2line_` 查找该地址在源代码中的位置和对应的函数名。

ESP-IDF 应用程序发生 crash 和 panic 事件时，将产生如下的寄存器转储和回溯：

```
Chapter 1. 快速入门

Guru Meditation Error of type StoreProhibited occurred on core 0. Exception was unhandled.
Register dump:
PC : 0x400f360d PS : 0x00060330 A0 : 0x800dbf56 A1 : ...
-0x3ffbe9e0
A2 : 0x3ffbe18c A3 : 0x00000005 A4 : 0x00000000 A5 : ...
-0x00000000
A6 : 0x00000000 A7 : 0x00000080 A8 : 0x00000000 A9 : ...
-0x3ffbe7d0
A10 : 0x00000003 A11 : 0x00060f23 A12 : 0x00060f20 A13 : ...
-0x3ffbe6d0
A14 : 0x00000047 A15 : 0x0000000f SAR : 0x00000019 EXCCAUSE:
-0x00000001d
EXCVADDR: 0x00000000 LBEG : 0x4000c46c LEND : 0x4000c477 LCOUNT :
-0x00000000
Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40...
-0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffbe90

IDF 监视器为寄存器转储如下信息:

Guru Meditation Error of type StoreProhibited occurred on core 0. Exception was unhandled.
Register dump:
PC : 0x400f360d PS : 0x00060330 A0 : 0x800dbf56 A1 : ...
-0x3ffbe9e0
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/. ./hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/. ./hello_world_main.c:52
A2 : 0x3ffbe18c A3 : 0x00000005 A4 : 0x00000000 A5 : ...
-0x00000000
A6 : 0x00000000 A7 : 0x00000080 A8 : 0x00000000 A9 : ...
-0x3ffbe7d0
A10 : 0x00000003 A11 : 0x00060f23 A12 : 0x00060f20 A13 : ...
-0x3ffbe6d0
A14 : 0x00000047 A15 : 0x0000000f SAR : 0x00000019 EXCCAUSE:
-0x00000001d
EXCVADDR: 0x00000000 LBEG : 0x4000c46c LEND : 0x4000c477 LCOUNT :
-0x00000000
Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40...
-0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffbe90
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/. ./hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/. ./hello_world_main.c:52
0x400dbf56: still_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/. ./hello_world_main.c:52
0x400dbf5e: dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/. ./hello_world_main.c:47
0x400dbf82: app_main at /home/gus/esp/32/idf/examples/get-started/hello_world/main/. ./hello_world_main.c:42
0x400d071d: main_task at /home/gus/esp/32/idf/examples/esp32/. ./cpu_start.c:254

IDF 监视器在后台运行以下命令，解码各地址:

xtensa-esp32-elf-addr2line -pfiaC -e build/PROJECT.elf ADDRESS

备注: 将环境变量 ESP_MONITOR_DECODE 设置为 0 或者调用 idf_monitor.py 的特定命令行选项
idf_monitor.py --disable-address-decoding 来禁止地址解码。

连接时复位目标芯片，默认情况下，IDF 监视器会在目标芯片连接时通过 DTR 和 RTS 串行线自动复位芯片。要防止 IDF 监视器在连接时自动复位，请在调用 IDF 监视器时加上选项 --no-reset，如 idf_monitor.py --no-reset。

备注：--no-reset 选项在 IDF 监视器连接到特定端口时可以实现同样的效果，如 idf.py monitor --no-reset -p [PORT]。

配置 GDBStub 以启用 GDB GDBStub 支持在运行时进行调试。GDBStub 在目标上运行，并通过串口连接到主机从而接收调试命令。GDBStub 支持读取内存和变量、检查调用栈帧等命令。虽然没有任何 JTAG 调试功能，但由于 GDBStub 完全通过串行端口完成通信，故不需要使用特殊硬件（如 JTAG/USB 桥接器）。

通过将 CONFIG_ESP_SYSTEM_PANIC 设置为 GDBStub on runtime，可以将目标配置为在后台运行 GDBStub。GDBStub 将保持在后台运行，直到通过串行端口发送 Ctrl+C 导致应用程序中断（即停止程序执行），从而让 GDBStub 处理调试命令。

此外，还可以通过设置 CONFIG_ESP_SYSTEM_PANIC 为 GDBStub on panic 来配置 panic 处理程序，使其在发生 crash 事件时运行 GDBStub。当 crash 发生时，GDBStub 将通过串口输出特殊的字符串模式，表示 GDBStub 正在运行。

无论是通过发送 Ctrl+C 还是收到特殊字符组合，IDF 监视器都会自动启动 GDB，从而让用户发送调试命令。GDB 退出后，通过 RTS 串口线复位目标。如果未连接 RTS 串口线，请按复位键，手动复位开发板。

备注：IDF 监视器在后台运行如下命令启用 GDB：

```
```

输出筛选  可以调用 idf.py monitor --print-filter="xyz" 启动 IDF 监视器，其中 --print-filter 是输出筛选的参数，参数默认为空字符串，可打印任何内容。

若需对打印内容设置限制，可指定 <tag>:<log_level> 等选项，其中 <tag> 是标签字符串，<log_level> 是 {N, E, W, I, D, V, *} 集合中的一个字母，指的是日志级别。

例如，PRINT_FILTER="tag1:*" 只匹配并打印 ESP_LOGW("tag1", ...) 所写的输出，或者写在较低日志详细级别输出的，即 ESP_LOGE("tag1", ...)。请勿指定 <log_level> 或使用详细级别默认值 *。

备注：编译时，可以使用主日志在 日志库 中禁用不必要的输出。也可以使用 IDF 监视器筛选输出来调整筛选设置，且无需重新编译应用程序。

应用程序筛选不能包含空格、星号 *、冒号 :，以便兼容输出筛选功能。

如果应用程序输出的最后一行后面没有回车，可能会影响输出筛选功能。即，监视器开始打印该行，但后来发现该行不应该被写入。这是一个已知问题，可以通过添加回车来避免此问题（特别是在没有输出紧跟其后的情况下）。

筛选规则示例

- * 可用于匹配任何类型标签。但 PRINT_FILTER="*:I tag1:E" 打印关于 tag1 的输出时会报错，这是因为 tag1 规则比 * 规则的优先级高。
• 默认规则（空）等价于 *:V，因为在详细级别或更高级别匹配任意标签即意味匹配所有内容。
• "*:N" 不仅抑制了日志功能的输出，也抑制了 printf 的打印输出。为了避免这一问题，请使用 ":E" 或更高的冗余级别。
• 规则 "tag1:V", "tag1:v", "tag1:" 和 "tag1:*" 同时。
• 规则 "tag1:W tag1:E" 等同于 "tag1:E"，这是因为后续出现的具有相同名称的标签会覆盖掉前一个标签。
• 规则 "tag1:I tag2:W" 仅在 Info 详细度级别或更高级别打印 tag1，在 Warning 详细度级别或更高级别打印 tag2。
• 规则 "tag1:W tag3:N" 本质上等同于一规则，这是因为 tag3:N 指定 tag3 不打印。
• tag3:N 在规则 "tag1:I tag2:W tag3:N *:V" 中更有意义，这是因为如果没有 tag3:N，tag3 信息就可能打印出来了；tag1 和 tag2 错误信息会打印在指定的详细度级别（或更高级别），并默认打印所有内容。

**高级筛选规则示例** 如下日志是在没有设置任何筛选选项的情况下获得的：

```
load:0x40078000, len:13564
entry 0x40078d4c
E (31) esp_image: image at 0x30000 has invalid magic byte
W (31) esp_image: image at 0x30000 has invalid SPI mode 255
E (39) boot: Factory app partition is not bootable
I (568) cpu_start: Pro cpu up.
I (569) heap_init: Initializing. RAM available for dynamic allocation:
I (603) cpu_start: Pro cpu start user code
D (309) light_driver: [light_init, 74]: status: 1, mode: 2
D (318) vfs: esp_vfs_register_fd_range is successful for range <54; 64) and VFS ID_ →1
I (328) wifi: wifi driver task: 3ffdf84, prio:23, stack:4096, core=0
```

PRINT_FILTER="wifi esp_image:E light_driver:I" 筛选选项捕获的输出如下所示：

```
E (31) esp_image: image at 0x30000 has invalid magic byte
I (328) wifi: wifi driver task: 3ffdf84, prio:23, stack:4096, core=0
```


```
load:0x40078000, len:13564
entry 0x40078d4c
I (569) heap_init: Initializing. RAM available for dynamic allocation:
D (309) light_driver: [light_init, 74]: status: 1, mode: 2
```

**IDF 监视器已知问题**

**Windows 环境下已知问题**

• 由于 Windows 控制台限制，有些箭头键及其他一些特殊键无法在 GDB 中使用。
• 偶然情况下，idf.py 退出时，可能会在 IDF 监视器恢复之前暂停 30 秒。
• GDB 运行时，可能会暂停一段时间，然后才开始与 GDBStub 进行通信。

**Linux 和 macOS 平台工具链的标准设置**

**详细安装步骤** 请根据下方详细步骤，完成安装过程。
设置开发环境 以下是为 ESP32 设置 ESP-IDF 的具体步骤。

- 第一步：安装准备
- 第二步：获取 ESP-IDF
- 第三步：设置工具
- 第四步：设置环境变量
- 第五步：开始使用 ESP-IDF 吧

第一步：安装准备 为了在 ESP32 中使用 ESP-IDF，需要根据操作系统安装一些软件包。以下安装指南可协助您安装 Linux 和 macOS 的系统上所有需要的软件包。

Linux 用户 编译 ESP-IDF 需要以下软件包，请根据使用的 Linux 发行版本，选择合适的安装命令。

- Ubuntu 和 Debian:
  ```bash
  sudo apt-get install git wget flex bison gperf python3 python3-venv python3-configure ninja-build ccache libffi-dev libssl1-dev dfu-util libusb-1.0-0-dev
  ```

- CentOS 7 & 8:
  ```bash
  sudo yum -y update && sudo yum install git wget flex bison gperf python3 python3-venv python3-configure ninja-build ccache dfu-util libusb-1.0-0-dev
  ```

目前仍然支持 CentOS 7，但为了更好的用户体验，建议使用 CentOS 8。

- Arch:
  ```bash
  sudo pacman -S --needed gcc git make flex bison gperf python cmake ninja-configure ccache dfu-util libusb
  ```

备注：
- 使用 ESP-IDF 需要 CMake 3.16 或以上版本。较早的 Linux 发行版可能需要升级自身的软件源仓库，或开启 backports 套件库，或安装“cmake3”软件包（不是安装“cmake”）。
- 如果上述列表中没有您使用的系统，请参考您所用系统的相关文档，查看安装软件包所用的命令。

macOS 用户 ESP-IDF 将使用 macOS 上默认安装的 Python 版本。

- 安装 CMake 和 Ninja 编译工具：
  - 若有 HomeBrew，您可以运行:
    ```bash
    brew install cmake ninja dfu-util
    ```
  - 若有 MacPorts，您可以运行:
    ```bash
    sudo port install cmake ninja dfu-util
    ```
  - 若以上均不适用，请访问 CMake 和 Ninja 主页，查询有关 macOS 平台的下载安装问题。
- 强烈建议同时安装 ccache 以获得更快的编译速度。如有 HomeBrew，可通过 MacPorts 上的 brew install ccache 或 sudo port install ccache 完成安装。

备注：如您在上述任何步骤中遇到以下错误:

```bash
xcrun: error: invalid active developer path (/Library/Developer/CommandLineTools), missing xcrun at /Library/Developer/CommandLineTools/usr/bin/xcrun
```

则必须安装 XCode 命令行工具，可运行 xcode-select --install 命令进行安装。
Apple M1 用户  如果您使用的是 Apple M1 系列且看到如下错误提示:

```
WARNING: directory for tool xtensa-esp32-elf version esp-2021r2-patch3-8.4.0 is present, but tool was not found
ERROR: tool xtensa-esp32-elf has no installed versions. Please run 'install.sh' to install it.
```

或者:

```
zsh: bad CPU type in executable: ~/.espressif/tools/xtensa-esp32-elf/esp-2021r2-patch3-8.4.0/xtensa-esp32-elf/bin/xtensa-esp32-elf-gcc
```

您需要运行如下命令来安装 Apple Rosetta 2:

```
/usr/sbin/softwareupdate --install-rosetta --agree-to-license
```

安装 Python 3  Catalina 10.15 发布说明 中表示不推荐使用 Python 2.7 版本，在未来的 macOS 版本中也不会默认包含 Python 2.7。执行以下命令来检查您当前使用的 Python 版本:

```
python --version
```

如果输出结果是 Python 2.7.17，则代表您的默认解析器是 Python 2.7。这时您需要运行以下命令检查电脑上是否已经安装过 Python 3:

```
python3 --version
```

如果运行上述命令出现错误，则代表电脑上没有安装 Python 3。

请根据以下步骤安装 Python 3:

- 使用 HomeBrew 进行安装的方法如下:
  ```
  brew install python3
  ```

- 使用 MacPorts 进行安装的方法如下:
  ```
  sudo port install python38
  ```

第二步：获取 ESP-IDF  在围绕 ESP32 构建应用程序之前，请先获取乐鑫提供的软件库文件 ESP-IDF 仓库。

获取 ESP-IDF 的本地副本：打开终端，切换到您要保存 ESP-IDF 的工作目录，使用 git clone 命令克隆远程仓库。针对不同操作系统的详细步骤，请见下文。

打开终端，运行以下命令:

```
mkdir -p ~/esp
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 将下载至 ~/esp/esp-idf。

请前往 ESP-IDF 版本简介，查看 ESP-IDF 不同版本的具体适用场景。

第三步：设置工具  除了 ESP-IDF 本身，您还需要为支持 ESP32 的项目安装 ESP-IDF 使用的各种工具，比如编译器、调试器、Python 包等。

```
cd ~/esp/esp-idf
./install.sh esp32
```

或使用 Fish shell:
上述命令仅为 ESP32 安装所需工具。如果需要为多个目标芯片开发项目，则可以一次性指定多个目标，如下所示:

```
cd ~/esp/esp-idf
./install.fish esp32
```

或使用 Fish shell:

```
cd ~/esp/esp-idf
./install.fish esp32,esp32s2
```

如果需要一次性为所有支持的目标芯片安装工具，可以运行如下命令:

```
cd ~/esp/esp-idf
./install.sh all
```

或使用 Fish shell:

```
cd ~/esp/esp-idf
./install.fish all
```

**备注：** 对于 macOS 用户，如您在上述任何步骤中遇到以下错误:

```
<urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable...
```

可运行您电脑 Python 文件夹中的 Install Certificates.command 安装证书。了解更多信息，请参考 安装 ESP-IDF 工具时出现的下载错误。

**下载工具备份方案** ESP-IDF 工具安装器会下载 Github 发布版本中附带的一些工具，如果访问 Github 较为缓慢，可以设置一个环境变量，从而优先选择 Espressif 的下载服务器进行 Github 资源下载。

**备注：** 该设置只影响从 Github 发布版本中下载的单个工具，它并不会改变访问任何 Git 仓库的 URL。

要在安装工具时优先选择 Espressif 下载服务器，请在运行 install.sh 时使用以下命令:

```
cd ~/esp/esp-idf
export IDF_GITHUB_ASSETS="dl.espressif.com/github_assets"
./install.sh
```

**自定义工具安装路径** 本步骤中介绍的脚本将 ESP-IDF 所需的编译工具默认安装在用户的根目录中，即 Linux 系统中的 $HOME/.espressif 目录。您可以选择将工具安装到其他目录中，但请在运行安装脚本前，重新设置环境变量 IDF_TOOLS_PATH。注意，请确保您的用户账号已经具备了读写该路径的权限。

如果修改了 IDF_TOOLS_PATH 变量，请确保该变量在每次执行安装脚本 (install.bat, install.psl 或 install.sh) 和导出脚本 (export.bat, export.psl 或 export.sh) 均保持一致。

**第四步：设置环境变量** 此时，您刚刚安装的工具尚未添加至 PATH 环境变量，无法通过“命令窗口”使用这些工具。因此，必须设置一些环境变量。这可以通过 ESP-IDF 提供的另一个脚本进行设置。

请在需要运行 ESP-IDF 的终端窗口运行以下命令:

```
Espressif Systems 116 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
```
对于 fish shell（仅支持 fish 3.0.0 及以上版本），请运行以下命令：

```
. $HOME/esp/esp-idf/export.fish
```

注意，命令开始的 “.” 与路径之间应有一个空格！

如果您需要经常运行 ESP-IDF，您可以为执行 export.sh 创建一个别名，具体步骤如下：

1. 复制并粘贴以下内容到 shell 配置文件中（.profile、.bashrc、.zprofile 等）
   ```
   alias get_idf=' . $HOME/esp/esp-idf/export.sh'
   ```

2. 通过重启终端窗口或运行 `source [path to profile]`，如 `source ~/.bashrc` 来刷新配置文件。

现在您可以在任何终端窗口中运行 `get_idf` 来设置或刷新 esp-idf 环境。

不建议直接将 export.sh 添加到 shell 的配置文件。这样做会导致在每个终端会话中都激活 IDF 虚拟环境（包括无需使用 IDF 的会话）。这违背了使用虚拟环境的目的，还可能影响其他软件的使用。

**第五步：开始使用 ESP-IDF** 现在您已经具备了使用 ESP-IDF 的所有条件，接下来将介绍如何开始您的第一个工程。

本指南将帮助您完成使用 ESP-IDF 的第一步。按照本指南，您将使用 ESP32 创建第一个工程，并构建、烧录和监控设备输出。

**备注：** 如果您还未安装 ESP-IDF，请参照安装 中的步骤，获取使用本指南所需的所有软件。

**开始创建工程** 现在，您可以准备开发 ESP32 应用程序了。您可以从 ESP-IDF 中 examples 目录下的 get-started/hello_world 工程开始。

**重要：** ESP-IDF 编译系统不支持 ESP-IDF 路径或其工程路径中带有空格。

将 get-started/hello_world 工程复制至您本地的 ~/esp 目录下：

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world .
```

**备注：** ESP-IDF 的 examples 目录下有一系列示例工程，您可以按照上述方法复制并运行其中的任何示例，也可以直接编译示例，无需进行复制。

**连接设备** 现在，请将您的 ESP32 开发板连接到 PC，并查看开发板使用的串口。

通常，串口在不同操作系统下显示的名称有所不同：

- **Linux 操作系统**：以 /dev/tty 开头
- **macOS 操作系统**：以 /dev/cu. 开头

有关如何查看串口名称的详细信息，请见与 ESP32 创建串口连接。

**备注：** 请记住串口号，您会在后续步骤中使用。
配置工程 请进入 hello_world 目录，设置 ESP32 为目标芯片，然后运行工程配置工具 menuconfig。

```
(cd ~/esp/hello_world
idf.py set-target esp32
idf.py menuconfig)
```

打开一个新工程后，应首先使用 idf.py set-target esp32 设置“目标”芯片。注意，此操作将清除并初始化项目之前的编译和配置（如有）。您也可以直接将“目标”配置为环境变量（此时可跳过该步骤）。更多信息，请见 Select the Target Chip: set-target。

正确操作上述步骤后，系统将显示以下菜单:

![工程配置菜单](image)

图 65: 工程配置—主窗口

您可以通过此菜单设置项目的具体变量，包括 Wi-Fi 网络名称、密码和处理器速度等。hello_world 示例项目会以默认配置运行，因此在这一项目中，可以跳过使用 menuconfig 进行项目配置这一步骤。

注意： 如果您使用的是 ESP32-DevKitC（板载 ESP32-SOLO-1 模组）或 ESP32-DevKitM-1（板载 ESP32-MINI-1(1U) 模组），请在烧写示例程序前，前往 menuconfig 中使能单核模式（CON-FIG_FREERTOS_UNICORE）。

备注： 您终端窗口中显示出的菜单颜色可能会与上图不同。您可以通过选项 --style 来改变外观。请运行 idf.py menuconfig --help 命令，获取更多信息。

如果您使用的是支持的开发板，可以通过板级支持包 (BSP) 来协助您的开发。更多信息，请见其他提示。

编译工程 请使用以下命令，编译烧录工程:

```
(idf.py build)
```

运行以上命令可以编译应用程序和所有 ESP-IDF 组件，接着生成引导加载程序、分区表和应用程序二进制文件。

```
$ idf.py build
Running cmake in directory /path/to/hello_world/build
Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"
```

(下页继续)
Warn about uninitialized values.
-- Found Git: /usr/bin/git (found version "2.17.0")
-- Building empty aws_iot component due to configuration
-- Component names: ...
-- Component paths: ...

... (more lines of build system output)

Generating hello_world.bin
esptool.py v2.3.1

Project build complete. To flash, run this command:

```
../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600 write_flash
--flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/hello_world.
--bin build 0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/
--partition-table.bin
```

or run 'idf.py -p PORT flash'

如果一切正常，编译完成后将生成.bin 文件。

**烧录到设备** 请运行以下命令，将刚刚生成的二进制文件烧录至您的 ESP32 开发板:

```
idf.py -p PORT flash
```

请将 PORT 替换为 ESP32 开发板的串口名称。如果 PORT 未定义，idf.py 将尝试使用可用的串口自动连接。

更多有关 idf.py 参数的详情，请见 idf.py。

**备注:** 勾选 flash 选项将自动编译并烧录工程，因此无需再运行 idf.py build。

若在烧录过程中遇到问题，请前往烧录故障排除 或与 ESP32 创建串口连接 获取更多详细信息。

**常规操作** 在烧录过程中，您会看到类似如下的输出日志:

```
...
esptool.py --chip esp32 -p /dev/ttyUSB0 -b 460800 --before=default_reset --
--after=hard_reset write_flash --flash_mode dio --flash_freq 40m --flash_size 2MB
--0x8000 partition_table/partition-table.bin 0x1000 bootloader/bootloader.bin
--0x10000 hello_world.bin
esptool.py v3.0-dev
Serial port /dev/ttyUSB0
Connecting........_
Chip is ESP32D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 40MHz
MAC: 24:0a:c4:05:b9:14
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds (effective 5962.8
kbit/s)...
Hash of data verified.
```
如果一切顺利，烧录完成后，开发板将会复位，应用程序“hello_world”开始运行。
如果您希望使用Eclipse或是VS Code IDE，而非idf.py，请参考Eclipse Plugin，以及VSCode Extension。

监视输出 您可以使用idf.py -p PORT monitor命令，监视“hello_world”工程的运行情况。注意，不要忘记将PORT替换为您的串口名称。

运行该命令后，IDF监视器应用程序将启动:

```
$ idf.py -p <PORT> monitor
Running idf_monitor in directory [...]/esp/hello_world/build
Executing "python [...]/esp-idf/tools/idf_monitor.py -b 115200 [...]/esp/hello_world/build/hello_world.elf"...
--- idf_monitor on <PORT> 115200 ---
--- Quit: Ctrl+I | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
```

此时，您就可以在启动日志和诊断日志之后，看到打印的“Hello world!”了。

```
Hello world!
... Restarting in 10 seconds...
... Restarting in 9 seconds...
Minimum free heap size: 298968 bytes
```

您可使用快捷键Ctrl+], 退出IDF监视器。
如果IDF监视器在烧录后很快就发生错误，或读取信息全是乱码（如下），很有可能是因为您的开发板采用了26 MHz 晶振，而ESP-IDF默认支持大多数开发板使用的40 MHz 晶振。
此时，您可以：

1. 退出监视器。
2. 返回 menuconfig。
3. 进入 Component config -> Hardware Settings -> Main XTAL Config -> Main XTAL frequency 进行配置，将 CONFIG_XTAL_FREQ_SEL 设置为 26 MHz。
4. 重新编译和烧录应用程序。

在当前的 ESP-IDF 版本中，ESP32 支持的主晶振频率如下：

- 26 MHz
- 40 MHz

备注：您也可以运行以下命令，一次性执行构建、烧录和监视过程：

```bash
idf.py -p PORT flash monitor
```

此外，

- 请前往 IDF 监视器，了解更多使用 IDF 监视器的快捷键和其他详情。
- 请前往 idf.py，查看更多 idf.py 命令和选项。

恭喜，您已完成 ESP32 的入门学习！现在，您可以尝试一些其他 examples，或者直接开发自己的应用程序。

重要：一些示例程序不支持 ESP32，因为 ESP32 中不包含所需的硬件。

在编译示例程序前请查看 README 文件中 Supported Targets 表格。如果表格中包含 ESP32，或者不存在这个表格，那么即表示 ESP32 支持这个示例程序。

其他提示

权限问题 /dev/ttyUSB0 使用某些 Linux 版本向 ESP32 烧录固件时，可能会出现 Failed to open port /dev/ttyUSB0 错误消息。此时可以将用户添加至 Linux Dialout 组。

兼容的 Python 版本 ESP-IDF 支持 Python 3.7 及以上版本，建议升级操作系统到最新版本从而更新 Python。也可选择从 sources 安装最新版 Python，或使用 Python 管理系统如 pyenv 对版本进行升级管理。

上手板级支持包 您可以使用 板级支持包 (BSP)，协助您在开发板上的原型开发。仅需要调用几个函数，便可以完成对特定开发板的初始化。

一般来说，BSP 支持开发板上所有硬件组件。除了管脚定义和初始化功能外，BSP 还附带如传感器、显示器、音频编解码器等外部元件的驱动程序。

BSP 通过 IDF 组件管理器 发布，您可以前往 IDF 组件注册器 进行下载。

以下示例演示了如何将 ESP-WROVER-KIT BSP 添加到项目中：

```bash
idf.py add-deperendency esp_wrover_kit
```

更多有关使用 BSP 的示例，请前往 BSP 示例文件夹。

擦除 flash ESP-IDF 支持擦除 flash。请运行以下命令，擦除整个 flash：
idf.py -p PORT erase-flash

若存在需要擦除的 OTA 数据，请运行以下命令:

idf.py -p PORT erase-otadata

擦除 flash 需要一段时间，在擦除过程中，请勿断开设备连接。

**建议：更新 ESP-IDF** 乐鑫会不时推出新版本的 ESP-IDF，修复 bug 或提供新的功能。请注意，EESP-IDF 的每个主要版本和次要版本都有相应的支持期限。支持期限满后，版本停止更新维护，用户可将项目升级到最新的 ESP-IDF 版本。更多关于支持期限的信息，请参考 ESP-IDF 版本。

因此，您在使用时，也应注意到更新您本地的版本。最简单的方法是：直接删除您本地的 esp-idf 文件夹，然后按照 第二步：获取 ESP-IDF 中的指示，重新完成克隆。

另一种方法是仅更新变更的部分。具体方式，请前往 更新 ESP-IDF 章节查看。具体更新步骤会根据您使用的 ESP-IDF 版本有所不同。

注意，更新完成后，请再次运行安装脚本，以防新版 ESP-IDF 所需的工具也有所更新。具体请参考 第三步：设置工具。

一旦重新安装好工具，请使用导出脚本更新环境，具体请参考 第四步：设置环境变量。

相关文档
- 与 ESP32 创建串口连接
- Eclipse Plugin
- VSCode Extension
- IDF 监视器

### 1.4 编译第一个工程

如果您已经在 ESP-IDF 且没有使用集成开发环境 (IDE)，请在命令提示行中按照在 Windows 中开始创建工程 或在 Linux 和 macOS 中开始创建工程 编译第一个工程。
Chapter 2

API 参考

2.1 API Conventions

This document describes conventions and assumptions common to ESP-IDF Application Programming Interfaces (APIs).

ESP-IDF provides several kinds of programming interfaces:

- C functions, structures, enums, type definitions and preprocessor macros declared in public header files of ESP-IDF components. Various pages in the API Reference section of the programming guide contain descriptions of these functions, structures and types.
- Build system functions, predefined variables and options. These are documented in the build system guide.
- Kconfig options can be used in code and in the build system (CMakeLists.txt) files.
- Host tools and their command line parameters are also part of ESP-IDF interface.

ESP-IDF consists of components written specifically for ESP-IDF as well as third-party libraries. In some cases, an ESP-IDF-specific wrapper is added to the third-party library, providing an interface that is either simpler or better integrated with the rest of ESP-IDF facilities. In other cases, the original API of the third-party library is presented to the application developers.

Following sections explain some of the aspects of ESP-IDF APIs and their usage.

2.1.1 Error handling

Most ESP-IDF APIs return error codes defined with esp_err_t type. See Error Handling section for more information about error handling approaches. Error Code Reference contains the list of error codes returned by ESP-IDF components.

2.1.2 Configuration structures

重要：Correct initialization of configuration structures is an important part in making the application compatible with future versions of ESP-IDF.

Most initialization or configuration functions in ESP-IDF take as an argument a pointer to a configuration structure. For example:
Initialization functions never store the pointer to the configuration structure, so it is safe to allocate the structure on
the stack.

The application must initialize all fields of the structure. The following is incorrect:

```c
esp_timer_create_args_t my_timer_args;
my_timer_args.callback = &my_timer_callback;
/* Incorrect! Fields .arg and .name are not initialized */
esp_timer_create(&my_timer_args, &my_timer);
```

Most ESP-IDF examples use C99 designated initializers for structure initialization, since they provide a concise way
of setting a subset of fields, and zero-initializing the remaining fields:

```c
const esp_timer_create_args_t my_timer_args = {
    .callback = &my_timer_callback,
    /* Correct, fields .arg and .name are zero-initialized */
};
```

C++ language doesn’t support the designated initializers syntax until C++20, however GCC compiler partially
supports it as an extension. When using ESP-IDF APIs in C++ code, you may consider using the following pattern:

```c
esp_timer_create_args_t my_timer_args = {};
/* All the fields are zero-initialized */
my_timer_args.callback = &my_timer_callback;
```

Default initializers

For some configuration structures, ESP-IDF provides macros for setting default values of fields:

```c
httpd_config_t config = HTTPD_DEFAULT_CONFIG();
/* HTTPD_DEFAULT_CONFIG expands to a designated initializer.
   Now all fields are set to the default values.
   Any field can still be modified: */
config.server_port = 8081;
httpd_handle_t server;
esp_err_t err = httpd_start(server, &config);
```

It is recommended to use default initializer macros whenever they are provided for a particular configuration structure.

### 2.1.3 Private APIs

Certain header files in ESP-IDF contain APIs intended to be used only in ESP-IDF source code, and not by the applica-
tions. Such header files often contain private or esp_private in their name or path. Certain components, such as hal
only contain private APIs.

Private APIs may be removed or changed in an incompatible way between minor or patch releases.

### 2.1.4 Components in example projects

ESP-IDF examples contain a variety of projects demonstrating usage of ESP-IDF APIs. In order to reduce code
duplication in the examples, a few common helpers are defined inside components that are used by multiple examples.
This includes components located in common_components directory, as well as some of the components located in the examples themselves. These components are not considered to be part of the ESP-IDF API.

It is not recommended to reference these components directly in custom projects (via EXTRA_COMPONENT_DIRS build system variable), as they may change significantly between ESP-IDF versions. When starting a new project based on an ESP-IDF example, copy both the project and the common components it depends on out of ESP-IDF, and treat the common components as part of the project. Note that the common components are written with examples in mind, and might not include all the error handling required for production applications. Take time to read the code and understand if it applicable to your use case.

2.1.5 API Stability

ESP-IDF uses Semantic Versioning as explained in the versions page.

Minor and bugfix releases of ESP-IDF guarantee compatibility with previous releases. The sections below explain different aspects and limitations to compatibility.

Source level compatibility

ESP-IDF guarantees source level compatibility of C functions, structures, enums, type definitions and preprocessor macros declared in public header files of ESP-IDF components. Source level compatibility implies that the application can be recompiled with the newer version of ESP-IDF without changes.

The following changes are allowed between minor versions and do not break source level compatibility:

- Deprecating functions (using the deprecated attribute) and header files (using a preprocessor #warning). Deprecations are listed in ESP-IDF release notes. It is recommended to update the source code to use the newer functions or files that replace the deprecated ones, however this is not mandatory. Deprecated functions and files can be removed in major versions of ESP-IDF.
- Renaming components, moving source and header files between components — provided that the build system ensures that correct files are still found.
- Renaming Kconfig options. Kconfig system renaming mechanism ensures that the original Kconfig option names can still be used by the application in sdkconfig file, CMake files and source code.

Lack of binary compatibility

ESP-IDF does not guarantee binary compatibility between releases. This means that if a precompiled library is built with one ESP-IDF version, it is not guaranteed to work the same way with the next minor or bugfix release. The following are the possible changes that keep source level compatibility but not binary compatibility:

- Changing numerical values for C enum members.
- Adding new structure members or changing the order of members. See Configuration structures for tips that help ensure compatibility.
- Replacing an extern function with a static inline one with the same signature, or vice versa.
- Replacing a function-like macro with a compatible C function.

Other exceptions from compatibility

While we try to make upgrading to a new ESP-IDF version easy, there are parts of ESP-IDF that may change between minor versions in an incompatible way. We appreciate issue reports about any unintended breaking changes that don’t fall into the categories below.

- Private APIs.
- Components in example projects.
- Features clearly marked as “beta”, “preview”, or “experimental”.
- Changes made to mitigate security issues or to replace insecure default behaviors with a secure ones.
• Features which were never functional. For example, if it was never possible to use a certain function or an enumeration value, it may get renamed (as part of fixing it) or removed. This includes software features which depend on non-functional chip hardware features.
• Unexpected or undefined behavior (for example, due to missing validation of argument ranges) that is not documented explicitly may be fixed/changed.
• Location of Kconfig options in menuconfig.
• Location and names of example projects.

2.2 应用层协议

2.2.1 ASIO port

Asio is a cross-platform C++ library, see https://think-async.com/Asio/. It provides a consistent asynchronous model using a modern C++ approach.

The ESP-IDF component ASIO has been moved from ESP-IDF since version v5.0 to a separate repository:

• ASIO component on GitHub

Hosted Documentation

The documentation can be found on the link below:

• ASIO documentation (English)

2.2.2 ESP-Modbus

The Espressif ESP-Modbus Library (esp-modbus) supports Modbus communication in the networks based on RS485, Wi-Fi, Ethernet interfaces. The ESP-IDF component freemodbus has been moved from ESP-IDF since version v5.0 to a separate repository:

• ESP-Modbus component on GitHub

Hosted Documentation

The documentation can be found on the link below:

• ESP-Modbus documentation (English)

Application Example

The examples below demonstrate the ESP-Modbus library of serial, TCP ports for slave and master implementations accordingly.

• protocols/modbus/serial/mb_slave
• protocols/modbus/serial/mb_master
• protocols/modbus/tcp/mb_tcp_slave
• protocols/modbus/tcp/mb_tcp_master

Please refer to the specific example README.md for details.
Protocol References


2.2.3 ESP-MQTT

概述

ESP-MQTT 是 MQTT 协议客户端的实现。MQTT 是一种基于发布/订阅模式的轻量级消息传输协议。

特性

- 支持基于 TCP 的 MQTT，基于 Mbed TLS 的 SSL、基于 WebSocket 的 MQTT 以及基于 WebSocket Secure 的 MQTT
- 通过 URI 简化配置流程
- 多个实例（一个应用程序中有多个客户端）
- 支持订阅、发布、认证、遗嘱消息，保持连接心跳机制以及 3 个服务质量 (QoS) 级别（组成全功能客户端）

应用示例

- protocols/mqtt/tcp: 基于 TCP 的 MQTT，默认端口 1883
- protocols/mqtt/tls: 基于 TLS 的 MQTT，默认端口 8883
- protocols/mqtt/tls_d: 基于 TLS 的 MQTT，使用数字签名外设进行身份验证，默认端口 8883
- protocols/mqtt/tls_mutual_auth: 基于 TLS 的 MQTT，使用证书进行身份验证，默认端口 8883
- protocols/mqtt/tls_psk: 基于 TLS 的 MQTT，使用预共享密钥进行身份验证，默认端口 8883
- protocols/mqtt/ws: 基于 WebSocket 的 MQTT，默认端口 80
- protocols/mqtt/wss: 基于 WebSocket Secure 的 MQTT，默认端口 443

配置

通过设置 esp_mqtt_client_config_t 结构体中的字段来进行配置。配置结构体包含以下子结构体，用于配置客户端的多种操作。

- broker - 允许设置地址和安全验证。
- credentials - 用于身份验证的客户端凭据。
- session - MQTT 会话相关配置。
- network - 网络相关配置。
- task - 允许配置 FreeRTOS 任务。
- buffer - 输入输出的缓冲区大小。

下文将详细介绍不同配置。

服务器

地址 通过 broker.address 结构体的 uri 字段或者 hostname, transport 以及 port 的组合，可以设置服务器地址。您也可以选择设置 path，该字段对 WebSocket 连接而言非常有用。

使用 uri 字段的格式为 scheme://hostname:port/path。

- 当前支持 mqtts, mqtt, ws 和 wss 协议
- 基于 TCP 的 MQTT 示例:
  - mqtts://mqtt.eclipseprojects.io: 基于 TCP 的 MQTT，默认端口 1883
  - mqtt://mqtt.eclipseprojects.io:1884: 基于 TCP 的 MQTT，端口 1884
为了验证服务器身份，对于使用 TLS 的安全连接，必须设置 `broker` 的 `verification` 结构体，服务器证书可设置为 PEM 或 DER 格式。如果选择 DER 格式，必须设置等长 `_len` 字段，否则应在 `certificate` 字段传入一个非空字符串的 PEM 格式字符串。

- 从服务器获取证书，例如：
  ```bash
  openssl s_client -connect mqtt.eclipseprojects.io:8883
  ```
- 检查示例应用程序：`examples/mqtt_ssl`
- 配置：

```c
const esp_mqtt_client_config_t mqtt_cfg = {
  .broker = {
    .address.uri = "mqtts://mqtt.eclipseprojects.io:8883",
  },
};
```
### 遗留消息 (LWT)
通过设置 `esp_mqtt_client_config_t.session.last_will` 结构体的以下字段，MQTT 会在一个客户端意外断开连接时通过遗留消息通知其他客户端。

- **topic**: 指向 LWT 消息主题的指针
- **msg**: 指向 LWT 消息的指针
- **msg_len**: LWT 消息的长度。msg 不可以为空字符结尾时需要该字段
- **qos**: LWT 消息的品质
- **retain**: 指定 LWT 消息的保留标识

### 在项目配置菜单中设置 MQTT
通过 `idf.py menuconfig`，可以在 Component config > ESP-MQTT Configuration 中找到 MQTT 设置。

相关设置如下:

- `CONFIG_MQTT_PROTOCOL_311`: 启用 MQTT 协议 3.1.1 版本
- `CONFIG_MQTT_TRANSPORT_SSL` 和 `CONFIG_MQTT_TRANSPORT_WEBSOCKET`: 启用特定 MQTT 传输层，例如 SSL, WEBSOCKET 和 WEBSOCKET_SECURE
- `CONFIG_MQTT_CUSTOM_OUTBOX`: 禁用 mqtt_outbox 默认实现，因此可以提供特定实现

### 事件

MQTT 客户端可能会发布以下事件:

- `MQTT_EVENT_BEFORE_CONNECT`: 客户端已初始化并即将开始连接至服务器。
- `MQTT_EVENT_CONNECTED`: 客户端已成功连接至服务器。客户端已准备好收发数据。
- `MQTT_EVENT_DISCONNECTED`: 由于无法读取或写入数据，例如因为服务器无法使用，客户端已终止连接。
- `MQTT_EVENT_SUBSCRIBED`: 服务器已确认客户端的订阅请求。事件数据将包含订阅消息的的消息 ID。
- `MQTT_EVENT_UNSUBSCRIBED`: 服务器已确认客户端的退订请求。事件数据将包含退订消息的消息 ID。
- `MQTT_EVENT_PUBLISHED`: 服务器已确认客户端的发布消息。消息将仅针对 QoS 级别 1 和 2 发布，因为级别 0 不会进行确认。事件数据将包含发布消息的消息 ID。
- `MQTT_EVENT_DATA`: 客户端已收到发布消息。事件数据包含：消息 ID, 发布消息所属主题名称, 收到的数据及其长度。对于超出内部缓冲区的数据，将发布多个 MQTT_EVENT_DATA，并更新事件数据的 current_data_offset 和 total_data_len 以跟踪碎片化消息。
- `MQTT_EVENT_ERROR`: 客户端遇到错误。使用事件数据 error_handle 中的 `esp_mqtt_error_type_t`，可以进一步判断错误类型。错误类型决定 error_handle 结构体的哪些部分会被填充。

### API 参考

#### Header File

- components/mqtt/esp-mqtt/include/mqtt_client.h

#### Functions

**esp_mqtt_client_handle_t esp_mqtt_client_init(const esp_mqtt_client_config_t *config)**

创建 MQTT 客户端 handle 基于配置。

参数 config - MQTT 配置结构体

返回 mqtt_client_handle 如果成功创建，NULL on error

**esp_err_t esp_mqtt_client_set_uri(esp_mqtt_client_handle_t client, const char *uri)**

设置 MQTT 连接 URI。此 API 通常用于覆盖在 esp_mqtt_client_init 中配置的 URI。

参数

- client - MQTT 客户端 handle
- uri -


**esp_err_t esp_mqtt_client_start (esp_mqtt_client_handle_t client)**

Starts MQTT client with already created client handle.

- **参数**: client - MQTT client handle
- **返回**: ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization ESP_FAIL on other error

**esp_err_t esp_mqtt_client_reconnect (esp_mqtt_client_handle_t client)**

This api is typically used to force reconnection upon a specific event.

- **参数**: client - MQTT client handle
- **返回**: ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization ESP_FAIL if client is in invalid state

**esp_err_t esp_mqtt_client_disconnect (esp_mqtt_client_handle_t client)**

This api is typically used to force disconnection from the broker.

- **参数**: client - MQTT client handle
- **返回**: ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization

**esp_err_t esp_mqtt_client_stop (esp_mqtt_client_handle_t client)**

Stops MQTT client tasks.

- **Notes**: Cannot be called from the MQTT event handler

**int esp_mqtt_client_subscribe (esp_mqtt_client_handle_t client, const char* topic, int qos)**

Subscribe the client to defined topic with defined qos.

- **Notes**: Cannot be called from the MQTT event handler

- **参数**: client - MQTT client handle, topic - defined topic, qos - defines quality of service
- **返回**: message_id of the subscribe message on success -1 on failure

**int esp_mqtt_client_unsubscribe (esp_mqtt_client_handle_t client, const char* topic)**

Unsubscribe the client from defined topic.

- **Notes**: Cannot be called from the MQTT event handler

- **参数**: client - MQTT client handle, topic - defined topic
- **返回**: message_id of the subscribe message on success -1 on failure
int esp_mqtt_client_publish (esp_mqtt_client_handle_t client, const char *topic, const char *data, int len, int qos, int retain)

Client to send a publish message to the broker.

Notes:
- This API might block for several seconds, either due to network timeout (10s) or if publishing payloads longer than internal buffer (due to message fragmentation)
- Client doesn’t have to be connected for this API to work, enqueueing the messages with qos>1 (returning -1 for all the qos=0 messages if disconnected). If MQTT_SKIP_PUBLISH_IF_DISCONNECTED is enabled, this API will not attempt to publish when the client is not connected and will always return -1.
- It is thread safe, please refer to esp_mqtt_client_subscribe for details.

参数
- client - MQTT client handle
- topic - topic string
- data - payload string (set to NULL, sending empty payload message)
- len - data length, if set to 0, length is calculated from payload string
- qos - QoS of publish message
- retain - retain flag

返回 message_id of the publish message (for QoS 0 message_id will always be zero) on success. -1 on failure.

int esp_mqtt_client_enqueue (esp_mqtt_client_handle_t client, const char *topic, const char *data, int len, int qos, int retain, bool store)

Enqueue a message to the outbox, to be sent later. Typically used for messages with qos>0, but could be also used for qos=0 messages if store=true.

This API generates and stores the publish message into the internal outbox and the actual sending to the network is performed in the mqtt-task context (in contrast to the esp_mqtt_client_publish() which sends the publish message immediately in the user task’s context). Thus, it could be used as a non blocking version of esp_mqtt_client_publish().

参数
- client - MQTT client handle
- topic - topic string
- data - payload string (set to NULL, sending empty payload message)
- len - data length, if set to 0, length is calculated from payload string
- qos - QoS of publish message
- retain - retain flag
- store - if true, all messages are enqueued; otherwise only QoS 1 and QoS 2 are enqueued

返回 message_id if queued successfully, -1 otherwise

esp_err_t esp_mqtt_client_destroy (esp_mqtt_client_handle_t client)

Destroys the client handle.

Notes:
- Cannot be called from the MQTT event handler

参数 client - MQTT client handle

返回 ESP_OK ESP_ERR_INVALID_ARG on wrong initialization

esp_err_t esp_mqtt_set_config (esp_mqtt_client_handle_t client, const esp_mqtt_client_config_t *config)

Set configuration structure, typically used when updating the config (i.e. on “before_connect” event.

参数
- client - MQTT client handle
- config - MQTT configuration structure

返回 ESP_ERR_NO_MEM if failed to allocate ESP_ERR_INVALID_ARG if conflicts on transport configuration. ESP_OK on success
**Register MQTT event.**

```c
esp_err_t esp_mqtt_client_register_event(esp_mqtt_client_handle_t client, esp_mqtt_event_id_t event, esp_event_handler_t event_handler, void *event_handler_arg)
```

Registers MQTT event.

- **client** - MQTT client handle
- **event** - event type
- **event_handler** - handler callback
- **event_handler_arg** - handlers context

Returns ESP_ERR_NO_MEM if failed to allocate ESP_ERR_INVALID_ARG on wrong initialization. ESP_OK on success.

```c
int esp_mqtt_client_get_outbox_size(esp_mqtt_client_handle_t client)
```

Get outbox size.

- **client** - MQTT client handle
- **outbox size** 0 on wrong initialization

**Structures**

```c
struct esp_mqtt_error_codes
```

MQTT error code structure to be passed as a contextual information into ERROR event

**Important:** This structure extends **esp_tls_last_error** error structure and is backward compatible with it (so might be down-casted and treated as **esp_tls_last_error** error, but recommended to update applications if used this way previously).

Use this structure directly checking **error_type** first and then appropriate error code depending on the source of the error:

<table>
<thead>
<tr>
<th>error_type</th>
<th>related member variables</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQTT_ERROR_TYPE_TCP_TRANSPORT</td>
<td>esp_tls_last_esp_err, esp_tls_stack_err, esp_tls_cert_verify_flags, sock_errno</td>
<td>Error reported from tcp_transport/esp-tls</td>
</tr>
<tr>
<td>MQTT_ERROR_TYPE_CONNECTION_REFUSED</td>
<td>connect_return_code</td>
<td>Internal error reported from MQTT broker on connection</td>
</tr>
</tbody>
</table>

**Public Members**

```c
esp_err_t esp_tls_last_esp_err
```

Last esp_err code reported from esp-tls component

```c
int esp_tls_stack_err
```

Tls specific error code reported from underlying tls stack

```c
int esp_tls_cert_verify_flags
```

Tls flags reported from underlying tls stack during certificate verification

```c
esp_mqtt_error_type_t error_type
```

Error type referring to the source of the error

```c
esp_mqtt_connect_return_code_t connect_return_code
```

Connection refused error code reported from MQTT broker on connection

```c
int esp_transport_sock_errno
```

Erno from the underlying socket
struct `esp_mqtt_event_t`  
`MQTT` event configuration structure

**Public Members**

`esp_mqtt_event_id_t` `event_id`  
`MQTT` event type

`esp_mqtt_client_handle_t` `client`  
`MQTT` client handle for this event

`char *data`  
Data associated with this event

`int data_len`  
Length of the data for this event

`int total_data_len`  
Total length of the data (longer data are supplied with multiple events)

`int current_data_offset`  
Actual offset for the data associated with this event

`char *topic`  
Topic associated with this event

`int topic_len`  
Length of the topic for this event associated with this event

`int msg_id`  
`MQTT` message id of message

`int session_present`  
`MQTT` session_present flag for connection event

`esp_mqtt_error_codes_t *error_handle`  
`esp-mqtt` error handle including esp-tls errors as well as internal `MQTT` errors

`bool retain`  
Retained flag of the message associated with this event

`int qos`  
QoS of the messages associated with this event

`bool dup`  
dup flag of the message associated with this event
exp_mqtt_protocol_ver_t protocol_ver

MQTT protocol version used for connection, defaults to value from menuconfig

struct esp_mqtt_client_config_t

MQTT client configuration structure

• Default values can be set via menuconfig
• All certificates and key data could be passed in PEM or DER format. PEM format must have a terminating NULL character and the related len field set to 0. DER format requires a related len field set to the correct length.

Public Members

struct esp_mqtt_client_config_t::broker_t broker

Broker address and security verification

struct esp_mqtt_client_config_t::credentials_t credentials

User credentials for broker

struct esp_mqtt_client_config_t::session_t session

MQTT session configuration.

struct esp_mqtt_client_config_t::network_t network

Network configuration

struct esp_mqtt_client_config_t::task_t task

FreeRTOS task configuration.

struct esp_mqtt_client_config_t::buffer_t buffer

Buffer size configuration.

struct broker_t

Broker related configuration

Public Members

struct esp_mqtt_client_config_t::broker_t::address_t address

Broker address configuration

struct esp_mqtt_client_config_t::broker_t::verification_t verification

Security verification of the broker

struct address_t

Broker address

• uri have precedence over other fields
• If uri isn’t set at least hostname, transport and port should.
Public Members

const char *uri
   Complete MQTT broker URI

const char *hostname
   Hostname, to set ipv4 pass it as string

esp_mqtt_transport_t transport
   Selects transport

const char *path
   Path in the URI

uint32_t port
   MQTT server port

struct verification_t
   Broker identity verification
   If fields are not set broker’s identity isn’t verified. It’s recommended to set the options in this struct for security reasons.

Public Members

bool use_global_ca_store
   Use a global ca_store, look esp-tls documentation for details.

esp_err_t (*crt_bundle_attach)(void *conf)
   Pointer to ESP x509 Certificate Bundle attach function for the usage of certificate bundles.

const char *certificate
   Certificate data, default is NULL, not required to verify the server.

size_t certificate_len
   Length of the buffer pointed to by certificate.

const struct psk_key_hint *psk_hint_key
   Pointer to PSK struct defined in esp_tls.h to enable PSK authentication (as alternative to certificate verification). PSK is enabled only if there are no other ways to verify broker.

bool skip_cert_common_name_check
   Skip any validation of server certificate CN field, this reduces the security of TLS and makes the MQTT client susceptible to MITM attacks

const char **alpn_protos
   NULL-terminated list of supported application protocols to be used for ALPN
struct buffer_t
Client buffer size configuration
Client have two buffers for input and output respectively.

Public Members

int size
size of MQTT send/receive buffer

int out_size
size of MQTT output buffer. If not defined, defaults to the size defined by buffer_size

struct credentials_t
Client related credentials for authentication.

Public Members

const char *username
MQTT username

const char *client_id
Set MQTT client identifier. Ignored if set_null_client_id == true If NULL set the default client id.
Default client id is ESP32_CHIPID% where CHIPID% are last 3 bytes of MAC address in hex format

bool set_null_client_id
Selects a NULL client id

struct esp_mqtt_client_config_t::credentials_t::authentication_t authentication
Client authentication

struct authentication_t
Client authentication
Fields related to client authentication by broker
For mutual authentication using TLS, user could select certificate and key, secure element or digital signature peripheral if available.

Public Members

const char *password
MQTT password

const char *certificate
Certificate for ssl mutual authentication, not required if mutual authentication is not needed.
Must be provided with key.
size_t certificate_len
Length of the buffer pointed to by certificate.

const char *key
Private key for SSL mutual authentication, not required if mutual authentication is not needed.
If it is not NULL, also certificate has to be provided.

size_t key_len
Length of the buffer pointed to by key.

const char *key_password
Client key decryption password, not PEM nor DER, if provided key_password_len must be correctly set.

int key_password_len
Length of the password pointed to by key_password

bool use_secure_element
Enable secure element, available in ESP32-ROOM-32SE, for SSL connection

void *ds_data
Carrier of handle for digital signature parameters, digital signature peripheral is available in some Espressif devices.

struct network_t
Network related configuration

Public Members

int reconnect_timeout_ms
Reconnect to the broker after this value in miliseconds if auto reconnect is not disabled (defaults to 10s)

int timeout_ms
Abort network operation if it is not completed after this value, in milliseconds (defaults to 10s).

int refresh_connection_after_ms
Refresh connection after this value (in milliseconds)

bool disable_auto_reconnect
Client will reconnect to server (when errors/disconnect). Set disable_auto_reconnect=true to disable

struct session_t
MQTT Session related configuration

Public Members
struct esp_mqtt_client_config_t::session_t::last_will_t last_will

Last will configuration

bool disable_clean_session

MQTT clean session, default clean_session is true

int keepalive

MQTT keepalive, default is 120 seconds

bool disable_keepalive

Set disable_keepalive=true to turn off keep-alive mechanism, keepalive is active by default. Note: setting the config value keepalive to 0 doesn’t disable keepalive feature, but uses a default keepalive period

esp_mqtt_protocol_ver_t protocol_ver

MQTT protocol version used for connection.

int message_retransmit_timeout

timeout for retransmitting of failed packet

struct last_will_t

Last Will and Testament message configuration.

Public Members

cost char *topic

LWT (Last Will and Testament) message topic

cost char *msg

LWT message, may be NULL terminated

int msg_len

LWT message length, if msg isn’t NULL terminated must have the correct length

int qos

LWT message QoS

int retain

LWT retained message flag

struct task_t

Client task configuration

Public Members

int priority

MQTT task priority
**int stack_size**

MQTT task stack size

**Macros**

**MQTT_ERROR_TYPE_ESP_TLS**

MQTT_ERROR_TYPE_TCP_TRANSPORT error type hold all sorts of transport layer errors, including ESP-TLS error, but in the past only the errors from MQTT_ERROR_TYPE_ESP_TLS layer were reported, so the ESP-TLS error type is re-defined here for backward compatibility.

**Type Definitions**

typedef struct esp_mqtt_client *esp_mqtt_client_handle_t

typedef enum esp_mqtt_event_id_t esp_mqtt_event_id_t

MQTT event types.

User event handler receives context data in esp_mqtt_event_t structure with

- client - MQTT client handle
- various other data depending on event type

typedef enum esp_mqtt_connect_return_code_t esp_mqtt_connect_return_code_t

MQTT connection error codes propagated via ERROR event.

typedef enum esp_mqtt_error_type_t esp_mqtt_error_type_t

MQTT connection error codes propagated via ERROR event.

typedef enum esp_mqtt_transport_t esp_mqtt_transport_t

MQTT protocol version used for connection.

typedef struct esp_mqtt_error_codes esp_mqtt_error_codes_t

MQTT error code structure to be passed as a contextual information into ERROR event.

Important: This structure extends esp_tls_last_error error structure and is backward compatible with it (so might be down-casted and treated as esp_tls_last_error error, but recommended to update applications if used this way previously).

Use this structure directly checking error_type first and then appropriate error code depending on the source of the error.

| error_type | related member variables | note | MQTT_ERROR_TYPE_TCP_TRANSPORT | esp_tls_last_error, esp_tls_stack_err, esp_tls_cert_verify_flags, sock_errno | Error reported from tcp_transport/esp-tls | MQTT_ERROR_TYPE_CONNECTION_REFUSED | connect_return_code | Internal error reported from MQTT broker on connection |

typedef struct esp_mqtt_event_t esp_mqtt_event_t

MQTT event configuration structure.

typedef esp_mqtt_event_t *esp_mqtt_event_handle_t
typedef esp_err_t (*mqtt_event_callback_t)(esp_mqtt_event_handle_t event)

typedef struct esp_mqtt_client_config_t esp_mqtt_client_config_t

 *MQTT* client configuration structure

  - Default values can be set via menuconfi
  - All certificates and key data could be passed in PEM or DER format. PEM format must have a terminating NULL character and the related len field set to 0. DER format requires a related len field set to the correct length.

**Enumerations**

enum esp_mqtt_event_id_t

 *MQTT* event types.

 User event handler receives context data in *esp_mqtt_event_t* structure with

  - *client* - *MQTT* client handle
  - various other data depending on event type

*Values*:

enumerator MQTT_EVENT_ANY

enumerator MQTT_EVENT_ERROR

  on error event, additional context: connection return code, error handle from esp_tls (if supported)

enumerator MQTT_EVENT_CONNECTED

  connected event, additional context: session_present flag

enumerator MQTT_EVENT_DISCONNECTED

  disconnected event

enumerator MQTT_EVENT_SUBSCRIBED

  subscribed event, additional context:

    - msg_id message id
    - data pointer to the received data
    - data_len length of the data for this event

'enumerator MQTT_EVENT_UNSUBSCRIBED

  unsubscribed event

enumerator MQTT_EVENT_PUBLISHED

  published event, additional context: msg_id

enumerator MQTT_EVENT_DATA

  data event, additional context:

    - msg_id message id
    - topic pointer to the received topic
    - topic_len length of the topic
    - data pointer to the received data
• data_len length of the data for this event
• current_data_offset offset of the current data for this event
• total_data_len total length of the data received
• retain retain flag of the message
• qos QoS level of the message
• dup dup flag of the message Note: Multiple MQTT_EVENT_DATA could be fired for one message, if it is longer than internal buffer. In that case only first event contains topic pointer and length, other contain data only with current data length and current data offset updating.

enumerator **MQTT_EVENT_BEFORE_CONNECT**
The event occurs before connecting

enumerator **MQTT_EVENT_DELETED**
Notification on delete of one message from the internal outbox, if the message couldn’t have been sent and acknowledged before expiring defined in OUTBOX_EXPIRED_TIMEOUT_MS. (events are not posted upon deletion of successfully acknowledged messages)
• This event id is posted only if MQTT_REPORT_DELETED_MESSAGES==1
• Additional context: msg_id (id of the deleted message).

enum **esp_mqtt_connect_return_code_t**
MQTT connection error codes propagated via ERROR event

Values:

enumerator **MQTT_CONNECTION_ACCEPTED**
Connection accepted

enumerator **MQTT_CONNECTION_REFUSE_PROTOCOL**
MQTT connection refused reason: Wrong protocol

enumerator **MQTT_CONNECTION_REFUSE_ID_REJECTED**
MQTT connection refused reason: ID rejected

enumerator **MQTT_CONNECTION_REFUSE_SERVER_UNAVAILABLE**
MQTT connection refused reason: Server unavailable

enumerator **MQTT_CONNECTION_REFUSE_BAD_USERNAME**
MQTT connection refused reason: Wrong user

enumerator **MQTT_CONNECTION_REFUSE_NOT_AUTHORIZED**
MQTT connection refused reason: Wrong username or password

enum **esp_mqtt_error_type_t**
MQTT connection error codes propagated via ERROR event

Values:

enumerator **MQTT_ERROR_TYPE_NONE**

enumerator **MQTT_ERROR_TYPE_TCP_TRANSPORT**

enumerator **MQTT_ERROR_TYPE_CONNECTION_REFUSED**
```c
enum esp_mqtt_transport_t
    Values:

    enumerator MQTT_TRANSPORT_UNKNOWN

    enumerator MQTT_TRANSPORT_OVER_TCP
        MQTT over TCP, using scheme: MQTT

    enumerator MQTT_TRANSPORT_OVER_SSL
        MQTT over SSL, using scheme: MQTTS

    enumerator MQTT_TRANSPORT_OVER_WS
        MQTT over Websocket, using scheme:: ws

    enumerator MQTT_TRANSPORT_OVER_WSS
        MQTT over Websocket Secure, using scheme: wss

enum esp_mqtt_protocol_ver_t
    MQTT protocol version used for connection
    Values:

    enumerator MQTT_PROTOCOL_UNDEFINED

    enumerator MQTT_PROTOCOL_V_3_1

    enumerator MQTT_PROTOCOL_V_3_1_1

    enumerator MQTT_PROTOCOL_V_5
```

### 2.2.4 ESP-TLS

#### Overview

The ESP-TLS component provides a simplified API interface for accessing the commonly used TLS functionality. It supports common scenarios like CA certification validation, SNI, ALPN negotiation, non-blocking connection among others. All the configuration can be specified in the `esp_tls_cfg_t` data structure. Once done, TLS communication can be conducted using the following APIs:

- `esp_tls_init()`: for initializing the TLS connection handle.
- `esp_tls_conn_new_sync()`: for opening a new blocking TLS connection.
- `esp_tls_conn_new_async()`: for opening a new non-blocking TLS connection.
- `esp_tls_conn_read()`: for reading from the connection.
- `esp_tls_conn_write()`: for writing into the connection.
- `esp_tls_conn_destroy()`: for freeing up the connection.

Any application layer protocol like HTTP1, HTTP2 etc can be executed on top of this layer.

#### Application Example

Simple HTTPS example that uses ESP-TLS to establish a secure socket connection: `protocols/https_request`.
Tree structure for ESP-TLS component

```
├── esp_tls.c
├── esp_tls.h
├── esp_tls_mbedtls.c
├── esp_tls_wolfssl.c
└── private_include
    ├── esp_tls_mbedtls.h
    └── esp_tls_wolfssl.h
```

The ESP-TLS component has a file `esp-tls/esp_tls.h` which contains the public API headers for the component. Internally ESP-TLS component uses one of the two SSL/TLS Libraries between mbedtls and wolfssl for its operation. API specific to mbedtls are present in `esp-tls/private_include/esp_tls_mbedtls.h` and API specific to wolfssl are present in `esp-tls/private_include/esp_tls_wolfssl.h`.

**TLS Server verification**

The ESP-TLS provides multiple options for TLS server verification on the client side. The ESP-TLS client can verify the server by validating the peer’s server certificate or with the help of pre-shared keys. The user should select only one of the following options in the `esp_tls_cfg_t` structure for TLS server verification. If no option is selected then client will return a fatal error by default at the time of the TLS connection setup.

- **cacert_buf and cacert_bytes**: The CA certificate can be provided in a buffer to the `esp_tls_cfg_t` structure. The ESP-TLS will use the CA certificate present in the buffer to verify the server. The following variables in `esp_tls_cfg_t` structure must be set.
  - `cacert_buf`: pointer to the buffer which contains the CA cert.
  - `cacert_bytes`: size of the CA certificate in bytes.

- **use_global_ca_store**: The `global_ca_store` can be initialized and set at once. Then it can be used to verify the server for all the ESP-TLS connections which have set `use_global_ca_store = true` in their respective `esp_tls_cfg_t` structure. See API Reference section below on information regarding different API used for initializing and setting up the `global_ca_store`.

- **crt_bundle_attach**: The ESP x509 Certificate Bundle API provides an easy way to include a bundle of custom x509 root certificates for TLS server verification. More details can be found at ESP x509 Certificate Bundle.

- **psk_hint_key**: To use pre-shared keys for server verification, `CONFIG_ESP_TLS_PSK_VERIFICATION` should be enabled in the ESP-TLS menuconfig. Then the pointer to PSK hint and key should be provided to the `esp_tls_cfg_t` structure. The ESP-TLS will use the PSK for server verification only when no other option regarding the server verification is selected.

- **skip server verification**: This is an insecure option provided in the ESP-TLS for testing purpose. The option can be set by enabling `CONFIG_ESP_TLS_INSECURE` and `CONFIG_ESP_TLS_SKIP_SERVER_CERT_VERIFY` in the ESP-TLS menuconfig. When this option is enabled the ESP-TLS will skip server verification by default when no other options for server verification are selected in the `esp_tls_cfg_t` structure. **WARNING:** Enabling this option comes with a potential risk of establishing a TLS connection with a server which has a fake identity, provided that the server certificate is not provided either through API or other mechanism like `ca_store` etc.

**ESP-TLS Server cert selection hook**

The ESP-TLS component provides an option to set the server cert selection hook when using the mbedTLS stack. This provides an ability to configure and use a certificate selection callback during server handshake, to select a certificate to present to the client based on the TLS extensions supplied in the client hello (alpn, sni, etc). To enable this feature, please enable `CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK` in the ESP-TLS menuconfig. The certificate selection callback can be configured in the `esp_tls_cfg_t` structure as follows:

```c
int cert_selection_callback(mbedtls_ssl_context *ssl)
{
    /* Code that the callback should execute */
    return 0;
}
```
esp_tls_cfg_t
cfg = {
    cert_select_cb = cert_section_callback,
};

Underlying SSL/TLS Library Options

The ESP-TLS component has an option to use mbedtls or wolfssl as their underlying SSL/TLS library. By default only mbedtls is available and is used, wolfssl SSL/TLS library is available publicly at https://github.com/espressif/esp-wolfssl. The repository provides wolfssl component in binary format, it also provides few examples which are useful for understanding the API. Please refer the repository README.md for information on licensing and other options. Please see below option for using wolfssl in your project.

注意事项: As the library options are internal to ESP-TLS, switching the libraries will not change ESP-TLS specific code for a project.

How to use wolfssl with ESP-IDF

There are two ways to use wolfssl in your project

1) Directly add wolfssl as a component in your project with following three commands:
   mkdir components
   cd components
   git clone https://github.com/espressif/esp-wolfssl.git

2) Add wolfssl as an extra component in your project.
   • Download wolfssl with:
     git clone https://github.com/espressif/esp-wolfssl.git
   • Include esp-wolfssl in ESP-IDF with setting EXTRA_COMPONENT_DIRS in CMakeLists.txt of your project as done in wolfssl/examples. For reference see Optional Project variables in build-system.

After above steps, you will have option to choose wolfssl as underlying SSL/TLS library in configuration menu of your project as follows:

```bash
idf.py menuconfig -> ESP-TLS -> choose SSL/TLS Library -> mbedtls/wolfssl
```

Comparison between mbedtls and wolfssl

The following table shows a typical comparison between wolfssl and mbedtls when protocols/https_request example (which has server authentication) was run with both SSL/TLS libraries and with all respective configurations set to default. (mbedtls IN_CONTENT length and OUT_CONTENT length were set to 16384 bytes and 4096 bytes respectively)

<table>
<thead>
<tr>
<th>Property</th>
<th>Wolfssl</th>
<th>Mbedtls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Heap Consumed</td>
<td>~19 Kb</td>
<td>~37 Kb</td>
</tr>
<tr>
<td>Task Stack Used</td>
<td>~2.2 Kb</td>
<td>~3.6 Kb</td>
</tr>
<tr>
<td>Bin size</td>
<td>~858 Kb</td>
<td>~736 Kb</td>
</tr>
</tbody>
</table>
ATECC608A (Secure Element) with ESP-TLS

ESP-TLS provides support for using ATECC608A cryptoauth chip with ESP32-WROOM-32SE. Use of ATECC608A is supported only when ESP-TLS is used with mbedTLS as its underlying SSL/TLS stack. ESP-TLS uses mbedtls as its underlying TLS/SSL stack by default unless changed manually.

To enable the secure element support, and use it in your project for TLS connection, you will have to follow below steps

1) Add esp-cryptoauthlib in your project, for details please refer esp-cryptoauthlib with ESP_IDF

2) Enable following menuconfig option:

```
menuconfig->Component config->ESP-TLS->Use Secure Element (ATECC608A) with ESP-TLS
```

3) Select type of ATECC608A chip with following option:

```
menuconfig->Component config->esp-cryptoauthlib->Choose Type of ATECC608A chip
```

to know more about different types of ATECC608A chips and how to obtain type of ATECC608A connected to your ESP module please visit [ATECC608A chip type]

4) Enable use of ATECC608A in ESP-TLS by providing following config option in esp_tls_cfg_t

```
esp_tls_cfg_t cfg = {
    /* other configurations options */
    .use_secure_element = true,
};
```

API Reference

**Header File**

- components/esp-tls/esp_tls.h

**Functions**

- `esp_tls_init` (void)
  
  Create TLS connection.
  
  This function allocates and initializes esp-tls structure handle.

- `esp_tls_conn_http_new` (const char *url, const esp_tls_cfg_t *cfg)
  
  Create a new blocking TLS/SSL connection with a given "HTTP" url.

  **Note**: This API is present for backward compatibility reasons. Alternative function with the same functionality is `esp_tls_conn_http_new_sync` (and its asynchronous version `esp_tls_conn_http_new_async`)

  **参数**
  
  - `url` - [in] url of host.
• **cfg** – [in] TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to ‘esp_tls_cfg_t’. At a minimum, this structure should be zero-initialized.

返回 pointer to esp_tls_t, or NULL if connection couldn’t be opened.

```c
int esp_tls_conn_new_sync(const char* hostname, int hostlen, int port, const esp_tls_cfg_t *cfg, esp_tls_t *tls)
```

Create a new blocking TLS/SSL connection.

This function establishes a TLS/SSL connection with the specified host in blocking manner.

参数

- `port` – [in] Port number of the host.
- `cfg` – [in] TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to esp_tls_cfg_t. At a minimum, this structure should be zero-initialized.
- `tls` – [in] Pointer to esp-tls as esp-tls handle.

返回

- -1 If connection establishment fails.
- 1 If connection establishment is successful.
- 0 If connection state is in progress.

```c
int esp_tls_conn_http_new_sync(const char* url, const esp_tls_cfg_t *cfg, esp_tls_t *tls)
```

Create a new blocking TLS/SSL connection with a given “HTTP” url.

The behaviour is same as esp_tls_conn_new_sync() API. However this API accepts host’s url.

参数

- `cfg` – [in] TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to ‘esp_tls_cfg_t’. At a minimum, this structure should be zero-initialized.
- `tls` – [in] Pointer to esp-tls as esp-tls handle.

返回

- -1 If connection establishment fails.
- 1 If connection establishment is successful.
- 0 If connection state is in progress.

```c
int esp_tls_conn_new_async(const char* hostname, int hostlen, int port, const esp_tls_cfg_t *cfg, esp_tls_t *tls)
```

Create a new non-blocking TLS/SSL connection.

This function initiates a non-blocking TLS/SSL connection with the specified host, but due to its non-blocking nature, it doesn’t wait for the connection to get established.

参数

- `port` – [in] Port number of the host.
- `cfg` – [in] TLS configuration as esp_tls_cfg_t. non_block member of this structure should be set to be true.
- `tls` – [in] pointer to esp-tls as esp-tls handle.

返回

- -1 If connection establishment fails.
- 0 If connection establishment is in progress.
- 1 If connection establishment is successful.

```c
int esp_tls_conn_http_new_async(const char* url, const esp_tls_cfg_t *cfg, esp_tls_t *tls)
```

Create a new non-blocking TLS/SSL connection with a given “HTTP” url.

The behaviour is same as esp_tls_conn_new_async() API. However this API accepts host’s url.
参数
  • `url` [in] url of host.
  • `cfg` [in] TLS configuration as esp_tls_cfg_t.
  • `tls` [in] pointer to esp-tls as esp-tls handle.

返回
  • -1 If connection establishment fails.
  • 0 If connection establishment is in progress.
  • 1 If connection establishment is successful.

`ssize_t esp_tls_conn_write (esp_tls_t *tls, const void *data, size_t datalen)`
Write from buffer ‘data’ into specified tls connection.

参数
  • `tls` [in] pointer to esp-tls as esp-tls handle.
  • `data` [in] Buffer from which data will be written.
  • `datalen` [in] Length of data buffer.

返回
  • >=0 if write operation was successful, the return value is the number of bytes actually
    written to the TLS/SSL connection.
  • <0 if write operation was not successful, because either an error occurred or an action must
    be taken by the calling process.
  • ESP_TLS_ERR_SSL_WANT_READ/ ESP_TLS_ERR_SSL_WANT_WRITE. if the
    handshake is incomplete and waiting for data to be available for reading. In this case this
    functions needs to be called again when the underlying transport is ready for operation.

`ssize_t esp_tls_conn_read (esp_tls_t *tls, void *data, size_t datalen)`
Read from specified tls connection into the buffer ‘data’.

参数
  • `tls` [in] pointer to esp-tls as esp-tls handle.
  • `data` [in] Buffer to hold read data.
  • `datalen` [in] Length of data buffer.

返回
  • >0 if read operation was successful, the return value is the number of bytes actually read
    from the TLS/SSL connection.
  • 0 if read operation was not successful. The underlying connection was closed.
  • <0 if read operation was not successful, because either an error occurred or an action must
    be taken by the calling process.

`int esp_tls_conn_destroy (esp_tls_t *tls)`
Close the TLS/SSL connection and free any allocated resources.

This function should be called to close each tls connection opened with `esp_tls_conn_new_sync()` (or
esp_tls_conn_http_new_sync()) and `esp_tls_conn_new_async()` (or esp_tls_conn_http_new_async()) APIs.

参数 `tls` [in] pointer to esp-tls as esp-tls handle.

返回 - 0 on success
  • -1 if socket error or an invalid argument

`ssize_t esp_tls_get_bytes_avail (esp_tls_t *tls)`
Return the number of application data bytes remaining to be read from the current record.

This API is a wrapper over mbdtls’ s mbdtls_ssl_get_bytes_avail() API.

参数 `tls` [in] pointer to esp-tls as esp-tls handle.

返回
  • -1 in case of invalid arg
  • bytes available in the application data record read buffer

`esp_err_t esp_tls_get_conn_sockfd (esp_tls_t *tls, int *sockfd)`
Returns the connection socket file descriptor from esp_tls session.

参数
- **tls** - [in] handle to esp_tls context
- **sockfd** - [out] int pointer to sockfd value.

*Return* - ESP_OK on success and value of sockfd will be updated with socket file descriptor for connection
  - ESP_ERR_INVALID_ARG if (tls == NULL || sockfd == NULL)

**void** esp_tls_get_ssl_context (esp_tls_t *tls)

Returns the ssl context.
- **tls** - [in] handle to esp_tls context
  - ssl_ctx pointer to ssl context of underlying TLS layer on success
  - NULL in case of error

**esp_err_t** esp_tls_init_global_ca_store (void)

Create a global CA store, initially empty.

This function should be called if the application wants to use the same CA store for multiple connections. This function initialises the global CA store which can be then set by calling esp_tls_set_global_ca_store(). To be effective, this function must be called before any call to esp_tls_set_global_ca_store().

*Return* - ESP_OK if creating global CA store was successful.
  - ESP_DS_NO_MEM if an error occurred when allocating the mbedtls resources.

**esp_err_t** esp_tls_set_global_ca_store (const unsigned char *cacert_pem_buf, const unsigned int cacert_pem_bytes)

Set the global CA store with the buffer provided in pem format.

This function should be called if the application wants to set the global CA store for multiple connections i.e. to add the certificates in the provided buffer to the certificate chain. This function implicitly calls esp_tls_init_global_ca_store() if it has not already been called. The application must call this function before calling esp_tls_conn_new().

- **cacert_pem_buf** - [in] Buffer which has certificates in pem format. This buffer is used for creating a global CA store, which can be used by other tls connections.
- **cacert_pem_bytes** - [in] Length of the buffer.

*Return* - ESP_OK if adding certificates was successful.
  - Other if an error occurred or an action must be taken by the calling process.

**void** esp_tls_free_global_ca_store (void)

Free the global CA store currently being used.

The memory being used by the global CA store to store all the parsed certificates is freed up. The application can call this API if it no longer needs the global CA store.

**esp_err_t** esp_tls_get_and_clear_last_error (esp_tls_error_handle_t h, int *esp_tls_code, int *esp_tls_flags)

Returns last error in esp_tls with detailed mbedtls related error codes. The error information is cleared internally upon return.

- **h** - [in] esp-tls error handle.
- **esp_tls_code** - [out] last error code returned from mbedtls api (set to zero if none)
  - This pointer could be NULL if caller does not care about esp_tls_code
- **esp_tls_flags** - [out] last certification verification flags (set to zero if none)
  - This pointer could be NULL if caller does not care about esp_tls_code

*Return* - ESP_ERR_INVALID_STATE if invalid parameters
  - ESP_OK (0) if no error occurred
  - specific error code (based on ESP_ERR_ESP_TLS_BASE) otherwise
**esp_err_t esp_tls_get_and_clear_error_type** *(esp_tls_error_handle_t h, esp_tls_error_type_t err_type, int *error_code)*

Returns the last error captured in esp_tls of a specific type. The error information is cleared internally upon return.

参数
- **h** – [in] esp-tls error handle.
- **err_type** – [in] specific error type
- **error_code** – [out] last error code returned from mbedtls api (set to zero if none) This pointer could be NULL if caller does not care about esp_tls_code

返回
- ESP_ERR_INVALID_STATE if invalid parameters
- ESP_OK if a valid error returned and was cleared

**esp_err_t esp_tls_get_error_handle** *(esp_tls_t *tls, esp_tls_error_handle_t *error_handle)*

Returns the ESP-TLS error_handle.

参数
- **tls** – [in] handle to esp_tls context
- **error_handle** – [out] pointer to the error handle.

返回
- ESP_OK on success and error_handle will be updated with the ESP-TLS error handle.
- ESP_ERR_INVALID_ARG if (tls == NULL || error_handle == NULL)

mbedtls_x509_crt* esp_tls_get_global_ca_store (void)

Get the pointer to the global CA store currently being used.

The application must first call esp_tls_set_global_ca_store(). Then the same CA store could be used by the application for APIs other than esp_tls.

备注： Modifying the pointer might cause a failure in verifying the certificates.

返回
- Pointer to the global CA store currently being used if successful.
- NULL if there is no global CA store set.

**esp_err_t esp_tls_plain_tcp_connect** *(const char *host, int hostlen, int port, const esp_tls_cfg_t *cfg, esp_tls_error_handle_t error_handle, int *sockfd)*

Creates a plain TCP connection, returning a valid socket fd on success or an error handle.

参数
- **host** – [in] Hostname of the host.
- **hostlen** – [in] Length of hostname.
- **port** – [in] Port number of the host.
- **cfg** – [in] ESP-TLS configuration as esp_tls_cfg_t.
- **error_handle** – [out] ESP-TLS error handle holding potential errors occurred during connection
- **sockfd** – [out] Socket descriptor if successfully connected on TCP layer

返回 ESP_OK on success ESP_ERR_INVALID_ARG if invalid output parameters ESP-TLS based error codes on failure

**Structures**

struct psk_key_hint

ESP-TLS preshared key and hint structure.
### Public Members

- **const uint8_t** *key*
  - key in PSK authentication mode in binary format

- **const size_t** *key_size*
  - length of the key

- **const char** *hint*
  - hint in PSK authentication mode in string format

- **struct tls_keep_alive_cfg**
  - esp-tls client session ticket ctx
  - Keep alive parameters structure

### Public Members

- **bool** *keep_alive_enable*
  - Enable keep-alive timeout

- **int** *keep_alive_idle*
  - Keep-alive idle time (second)

- **int** *keep_alive_interval*
  - Keep-alive interval time (second)

- **int** *keep_alive_count*
  - Keep-alive packet retry send count

- **struct esp_tls_cfg**
  - ESP-TLS configuration parameters.

---

**备注**: Note about format of certificates:

- This structure includes certificates of a Certificate Authority, of client or server as well as private keys, which may be of PEM or DER format. In case of PEM format, the buffer must be NULL terminated (with NULL character included in certificate size).

- Certificate Authority’s certificate may be a chain of certificates in case of PEM format, but could be only one certificate in case of DER format.

- Variables names of certificates and private key buffers and sizes are defined as unions providing backward compatibility for legacy *_pem_buf and *_pem_bytes names which suggested only PEM format was supported. It is encouraged to use generic names such as cacert_buf and cacert_bytes.

---

### Public Members

- **const char** **alpn_protos**
  - Application protocols required for HTTP2. If HTTP2/ALPN support is required, a list of protocols that should be negotiated. The format is length followed by protocol name. For the most common cases the following is ok: const char **alpn_protos = [ “h2” , NULL ];**
where ‘h2’ is the protocol name

c = const unsigned char *cacert_buf
    Certificate Authority’s certificate in a buffer. Format may be PEM or DER, depending on mbedtls-support. This buffer should be NULL terminated in case of PEM

c = const unsigned char *cacert_pem_buf
    CA certificate buffer legacy name

unsigned int cacert_bytes
    Size of Certificate Authority certificate pointed to by cacert_buf (including NULL-terminator in case of PEM format)

unsigned int cacert_pem_bytes
    Size of Certificate Authority certificate legacy name

c = const unsigned char *clientcert_buf
    Client certificate in a buffer. Format may be PEM or DER, depending on mbedtls-support. This buffer should be NULL terminated in case of PEM

c = const unsigned char *clientcert_pem_buf
    Client certificate legacy name

unsigned int clientcert_bytes
    Size of client certificate pointed to by clientcert_pem_buf (including NULL-terminator in case of PEM format)

unsigned int clientcert_pem_bytes
    Size of client certificate legacy name

c = const unsigned char *clientkey_buf
    Client key in a buffer. Format may be PEM or DER, depending on mbedtls-support. This buffer should be NULL terminated in case of PEM

c = const unsigned char *clientkey_pem_buf
    Client key legacy name

unsigned int clientkey_bytes
    Size of client key pointed to by clientkey_pem_buf (including NULL-terminator in case of PEM format)

unsigned int clientkey_pem_bytes
    Size of client key legacy name

const unsigned char *clientkey_password
    Client key decryption password string

unsigned int clientkey_password_len
    String length of the password pointed to by clientkey_password
bool non_block
Configure non-blocking mode. If set to true the underneath socket will be configured in non blocking mode after TLS session is established.

bool use_secure_element
Enable this option to use secure element or atcc608a chip (Integrated with ESP32-WROOM-32SE)

int timeout_ms
Network timeout in milliseconds. Note: If this value is not set, by default the timeout is set to 10 seconds. If you wish that the session should wait indefinitely then please use a larger value e.g., INT32_MAX

bool use_global_ca_store
Use a global ca_store for all the connections in which this bool is set.

const char *common_name
If non-NULL, server certificate CN must match this name. If NULL, server certificate CN must match hostname.

bool skip_common_name
Skip any validation of server certificate CN field

tls_keep_alive_cfg_t *keep_alive_cfg
Enable TCP keep-alive timeout for SSL connection

const psk_hint_key_t *psk_hint_key
Pointer to PSK hint and key. if not NULL (and certificates are NULL) then PSK authentication is enabled with configured setup. Important note: the pointer must be valid for connection

esp_err_t (*crt_bundle_attach)(void *conf)
Function pointer to esp_crt_bundle_attach. Enables the use of certification bundle for server verification, must be enabled in menuconfig

void *ds_data
Pointer for digital signature peripheral context

bool is_plain_tcp
Use non-TLS connection: When set to true, the esp-tls uses plain TCP transport rather then TLS/SSL connection. Note, that it is possible to connect using a plain tcp transport directly with esp_tls_plain_tcp_connect() API

struct ifreq *if_name
The name of interface for data to go through. Use the default interface without setting

Type Definitions

typedef enum esp_tls_conn_state esp_tls_conn_state_t
ESP-TLS Connection State.

typedef enum esp_tls_role esp_tls_role_t
typedef struct psk_key_hint psk_hint_key_t
    ESP-TLS preshared key and hint structure.

typedef struct tls_keep_alive_cfg tls_keep_alive_cfg_t
    esp-tls client session ticket ctx
    Keep alive parameters structure

typedef struct esp_tls_cfg esp_tls_cfg_t
    ESP-TLS configuration parameters.

Note about format of certificates:

• This structure includes certificates of a Certificate Authority, of client or server as well as private keys, which may be of PEM or DER format. In case of PEM format, the buffer must be NULL terminated (with NULL character included in certificate size).
• Certificate Authority’s certificate may be a chain of certificates in case of PEM format, but could be only one certificate in case of DER format
• Variables names of certificates and private key buffers and sizes are defined as unions providing backward compatibility for legacy *_pem_buf and *_pem_bytes names which suggested only PEM format was supported. It is encouraged to use generic names such as cacert_buf and cacert_bytes.

typedef struct esp_tls esp_tls_t

Enumerations

enum esp_tls_conn_state
    ESP-TLS Connection State.
    Values:
    enumerator ESP_TLS_INIT
    enumerator ESP_TLS_CONNECTING
    enumerator ESP_TLS_HANDSHAKE
    enumerator ESP_TLS_FAIL
    enumerator ESP_TLS_DONE

enum esp_tls_role
    Values:
    enumerator ESP_TLS_CLIENT
    enumerator ESP_TLS_SERVER

Header File

• components/esp-tls/esp_tls_errors.h
Chapter 2. API 参考

Structures

struct **esp_tls_last_error**
Error structure containing relevant errors in case tls error occurred.

**Public Members**

```c
enum esp_err_t last_error
   error code (based on ESP_ERR_ESP_TLS_BASE) of the last occurred error

int esp_tls_error_code
   esp_tls error code from last esp_tls failed api

int esp_tls_flags
   last certification verification flags
```

**Macros**

```c
#define ESP_ERR_ESP_TLS_BASE
   Starting number of ESP-TLS error codes

#define ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME
   Error if hostname couldn’t be resolved upon tls connection

#define ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET
   Failed to create socket

#define ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY
   Unsupported protocol family

#define ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST
   Failed to connect to host

#define ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED
   failed to set/get socket option

#define ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT
   new connection in esp_tls_low_level_conn connection timed out

#define ESP_ERR_ESP_TLS_SE_FAILED

#define ESP_ERR_ESP_TLS_TCP_CLOSED_FIN

#define ESP_ERR_MBEDTLS_CERT_PARTLY_OK
   mbedtls parse certificates was partly successful

#define ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED
   mbedtls api returned error
```
ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED
  mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED
  mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED
  mbedtls api returned error

ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED
  mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_own_CERT_FAILED
  mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SETUP_FAILED
  mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_WRITE_FAILED
  mbedtls api returned error

ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED
  mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED
  mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED
  mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_TICKET_SETUP_FAILED
  mbedtls api returned failed

ESP_ERR_WOLFSSL_SSL_SET_HOSTNAME_FAILED
  wolfSSL api returned error

ESP_ERR_WOLFSSL_SSL_CONF_ALPN_PROTOCOLS_FAILED
  wolfSSL api returned error

ESP_ERR_WOLFSSL_CERT_VERIFY_SETUP_FAILED
  wolfSSL api returned error

ESP_ERR_WOLFSSL_KEY_VERIFY_SETUP_FAILED
  wolfSSL api returned error

ESP_ERR_WOLFSSL_SSL_HANDSHAKE_FAILED
  wolfSSL api returned failed
Chapter 2. API 参考

ESP_ERR_WOLFSSL_CTX_SETUP_FAILED
wolfSSL api returned failed

ESP_ERR_WOLFSSL_SSL_SETUP_FAILED
wolfSSL api returned failed

ESP_ERR_WOLFSSL_SSL_WRITE_FAILED
wolfSSL api returned failed

ESP_TLS_ERR_SSL_WANT_READ
Definition of errors reported from IO API (potentially non-blocking) in case of error:
  • esp_tls_conn_read()
  • esp_tls_conn_write()

ESP_TLS_ERR_SSL_WANT_WRITE

ESP_TLS_ERR_SSL_TIMEOUT

Type Definitions

typedef struct esp_tls_last_error *esp_tls_error_handle_t

typedef struct esp_tls_last_error esp_tls_last_error_t
  Error structure containing relevant errors in case tls error occurred.

Enumerations

denum esp_tls_error_type_t
  Definition of different types/sources of error codes reported from different components
  Values:

  enumerator ESP_TLS_ERR_TYPE_UNKNOWN

  enumerator ESP_TLS_ERR_TYPE_SYSTEM
    System error &#8212; errno

  enumerator ESP_TLS_ERR_TYPE_MBEDTLS
    Error code from mbedTLS library

  enumerator ESP_TLS_ERR_TYPE_MBEDTLS_CERT_FLAGS
    Certificate flags defined in mbedTLS

  enumerator ESP_TLS_ERR_TYPE_ESP
    ESP-IDF error type &#8212; esp_err_t

  enumerator ESP_TLS_ERR_TYPE_WOLFSSL
    Error code from wolfSSL library
Chapter 2. API

enumerator **ESP_TLS_ERR_TYPE_WOLFSSL_CERT_FLAGS**
Certificate flags defined in wolfSSL

enumerator **ESP_TLS_ERR_TYPE_MAX**
Last err type &#8212; invalid entry

### 2.2.5 ESP HTTP Client

#### Overview

`esp_http_client` provides an API for making HTTP/S requests from ESP-IDF applications. The steps to use this API are as follows:

- **`esp_http_client_init()`**: Creates an `esp_http_client_handle_t` instance i.e. a HTTP client handle based on the given `esp_http_client_config_t` configuration. This function must be the first to be called; default values will be assumed for the configuration values that are not explicitly defined by the user.
- **`esp_http_client_perform()`**: Performs all operations of the `esp_http_client` - opening the connection, exchanging data and closing the connection (as required), while blocking the current task until its completion. All related events will be invoked through the event handler (as specified in `esp_http_client_config_t`).
- **`esp_http_client_cleanup()`**: Closes the connection (if any) and frees up all the memory allocated to the HTTP client instance. This must be the last function to be called after the completion of operations.

#### Application Example

Simple example that uses ESP HTTP Client to make HTTP/S requests at protocols/esp_http_client.

#### Basic HTTP request

Check out the example functions `http_rest_with_url` and `http_rest_with_hostname_path` in the application example for implementation details.

#### Persistent Connections

Persistent connection means that the HTTP client can re-use the same connection for several exchanges. If the server does not request to close the connection with the `Connection: close` header, the connection is not dropped but is instead kept open and used for further requests.

To allow ESP HTTP client to take full advantage of persistent connections, one should make as many requests as possible using the same handle instance.

Check out the example functions `http_rest_with_url` and `http_rest_with_hostname_path` in the application example. Here, once the connection is created, multiple requests (GET, POST, PUT, etc.) are made before the connection is closed.

**Use Secure Element (ATECC608) for TLS**

A secure element (ATECC608) can be also used for the underlying TLS connection in the HTTP client connection. Please refer to Secure Element with ESP-TLS section in the `doc:` ESP-TLS documentation</api-reference/protocols/esp_tls` for more details. The secure element support has to be first enabled in menuconfig through `CONFIG_ESP_TLS_USE_SECURE_ELEMENT`. Then the HTTP client can be configured to use secure element as follows:
```c
esp_http_client_config_t cfg = {
    /* other configurations options */
    .use_secure_element = true,
};
```

## HTTPS Request

ESP HTTP client supports SSL connections using mbedTLS, with the `url` configuration starting with `https` scheme or `transport_type` set to `HTTP_TRANSPORT_OVER_SSL`. HTTPS support can be configured via `CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS` (enabled by default).

**Note:** While making HTTPS requests, if server verification is needed, additional root certificate (in PEM format) needs to be provided to the `cert_pem` member in `esp_http_client_config_t` configuration. Users can also use the ESP x509 Certificate Bundle for server verification using the `crt_bundle_attach` member of the `esp_http_client_config_t` configuration.

Check out the example functions `https_with_url` and `https_with_hostname_path` in the application example. (Implementation details of the above note are found here)

## HTTP Stream

Some applications need to open the connection and control the exchange of data actively (data streaming). In such cases, the application flow is different from regular requests. Example flow is given below:

- `esp_http_client_init()`: Create a HTTP client handle
- `esp_http_client_set_*` or `esp_http_client_delete_*`: Modify the HTTP connection parameters (optional)
- `esp_http_client_open()`: Open the HTTP connection with `write_len` parameter (content length that needs to be written to server), set `write_len=0` for read-only connection
- `esp_http_client_write()`: Write data to server with a maximum length equal to `write_len` of `esp_http_client_open()` function; no need to call this function for `write_len=0`
- `esp_http_client_fetch_headers()`: Read the HTTP Server response headers, after sending the request headers and server data (if any). Returns the content-length from the server and can be succeeded by `esp_http_client_get_status_code()` for getting the HTTP status of the connection.
- `esp_http_client_read()`: Read the HTTP stream
- `esp_http_client_close()`: Close the connection
- `esp_http_client_cleanup()`: Release allocated resources

Check out the example function `http_perform_as_stream_reader` in the application example for implementation details.

## HTTP Authentication

**ESP HTTP client supports both Basic and Digest Authentication.**

- Users can provide the username and password in the `url` or the `username` and `password` members of the `esp_http_client_config_t` configuration. For `auth_type = HTTP_AUTH_TYPE_BASIC`, the HTTP client takes only 1 perform operation to pass the authentication process.
- If `auth_type = HTTP_AUTH_TYPE_NONE`, but the `username` and `password` fields are present in the configuration, the HTTP client takes 2 perform operations. The client will receive the 401 Unauthorized header in its first attempt to connect to the server. Based on this information, it decides which authentication method to choose and performs it in the second operation.

Check out the example functions `http_auth_basic`, `http_auth_basic_redirect` (for Basic authentication) and `http_auth_digest` (for Digest authentication) in the application example for implementation details.
Examples of Authentication Configuration

- Authentication with URI

```c
esp_http_client_config_t config = {
    .url = "http://user:passwd@httpbin.org/basic-auth/user/passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

- Authentication with username and password entry

```c
esp_http_client_config_t config = {
    .url = "http://httpbin.org/basic-auth/user/passwd",
    .username = "user",
    .password = "passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

API Reference

Header File

- components/esp_http_client/include/esp_http_client.h

Functions

`esp_http_client_handle_t esp_http_client_init` (const `esp_http_client_config_t` *config)

Start a HTTP session This function must be the first function to call, and it returns a `esp_http_client_handle_t` that you must use as input to other functions in the interface. This call MUST have a corresponding call to `esp_http_client_cleanup` when the operation is complete.

参数 config - [in] The configurations, see `http_client_config_t`

返回

- `esp_http_client_handle_t`
- `NULL` if any errors

`esp_err_t esp_http_client_perform` (esp_http_client_handle_t client)

Invoke this function after `esp_http_client_init` and all the options calls are made, and will perform the transfer as described in the options. It must be called with the same `esp_http_client_handle_t` as input as the `esp_http_client_init` call returned. `esp_http_client_perform` performs the entire request in either blocking or non-blocking manner. By default, the API performs request in a blocking manner and returns when done, or if it failed, and in non-blocking manner, it returns if EAGAIN/EWOULDBLOCK or EINPROGRESS is encountered, or if it failed. And in case of non-blocking request, the user may call this API multiple times unless request & response is complete or there is a failure. To enable non-blocking `esp_http_client_perform()`, `is_async` member of `esp_http_client_config_t` must be set while making a call to `esp_http_client_init()` API. You can do any amount of calls to `esp_http_client_perform` while using the same `esp_http_client_handle_t`. The underlying connection may be kept open if the server allows it. If you intend to transfer more than one file, you are even encouraged to do so. `esp_http_client` will then attempt to re-use the same connection for the following transfers, thus making the operations faster, less CPU intense and using less network resources. Just note that you will have to use `esp_http_client_set_*` between the invoke to set options for the following `esp_http_client_perform`.

备注： You must never call this function simultaneously from two places using the same client handle. Let the function return first before invoking it another time. If you want parallel transfers, you must use several `esp_http_client_handle_t`. This function include `esp_http_client_open` -> `esp_http_client_write` -> `esp_http_client_fetch_headers` -> `esp_http_client_read` (and option) `esp_http_client_close`.

参数 client - The `esp_http_client` handle
**esp_err_t esp_http_client_set_url** (esp_http_client_handle_t client, const char *url)

Set URL for client, when performing this behavior, the options in the URL will replace the old ones.

- **client** - [in] The esp_http_client handle
- **url** - [in] The url

**Return**
- ESP_OK
- ESP_FAIL

**esp_err_t esp_http_client_set_post_field** (esp_http_client_handle_t client, const char *data, int len)

Set post data, this function must be called before esp_http_client_perform. Note: The data parameter passed to this function is a pointer and this function will not copy the data.

- **client** - [in] The esp_http_client handle
- **data** - [in] post data pointer
- **len** - [in] post length

**Return**
- ESP_OK
- ESP_FAIL

**int esp_http_client_get_post_field** (esp_http_client_handle_t client, char **data)

Get current post field information.

- **client** - [in] The esp_http_client handle
- **data** - [out] Point to post data pointer

**Return**
Size of post data

**esp_err_t esp_http_client_set_header** (esp_http_client_handle_t client, const char *key, const char *value)

Set http request header, this function must be called after esp_http_client_init and before any perform function.

- **client** - [in] The esp_http_client handle
- **key** - [in] The header key
- **value** - [in] The header value

**Return**
- ESP_OK
- ESP_FAIL

**esp_err_t esp_http_client_get_header** (esp_http_client_handle_t client, const char *key, char **value)

Get http request header. The value parameter will be set to NULL if there is no header which is same as the key specified, otherwise the address of header value will be assigned to value parameter. This function must be called after esp_http_client_init.

- **client** - [in] The esp_http_client handle
- **key** - [in] The header key
- **value** - [out] The header value

**Return**
- ESP_OK
- ESP_FAIL

**esp_err_t esp_http_client_get_username** (esp_http_client_handle_t client, char **value)

Get http request username. The address of username buffer will be assigned to value parameter. This function must be called after esp_http_client_init.

- **client** - [in] The esp_http_client handle
- **key** - [in] The header key
- **value** - [out] The header value

**Return**
- ESP_OK
- ESP_FAIL
### API Reference

**esp_err_t esp_http_client_set_username (esp_http_client_handle_t client, const char* username)

Set http request username. The value of username parameter will be assigned to username buffer. If the username parameter is NULL then username buffer will be freed.

- **Parameters**
  - `client` - [in] The esp_http_client handle
  - `username` - [in] The username value

- **Return**
  - ESP_OK
  - ESP_ERR_INVALID_ARG

**esp_err_t esp_http_client_get_password (esp_http_client_handle_t client, char** value)

Get http request password. The address of password buffer will be assigned to value parameter. This function must be called after esp_http_client_init.

- **Parameters**
  - `client` - [in] The esp_http_client handle
  - `value` - [out] The password value

- **Return**
  - ESP_OK
  - ESP_ERR_INVALID_ARG

**esp_err_t esp_http_client_set_password (esp_http_client_handle_t client, const char* password)

Set http request password. The value of password parameter will be assigned to password buffer. If the password parameter is NULL then password buffer will be freed.

- **Parameters**
  - `client` - [in] The esp_http_client handle
  - `password` - [in] The password value

- **Return**
  - ESP_OK
  - ESP_ERR_INVALID_ARG

**esp_err_t esp_http_client_set_auth_type (esp_http_client_handle_t client, esp_http_client_auth_type_t auth_type)

Set http request auth_type.

- **Parameters**
  - `client` - [in] The esp_http_client handle
  - `auth_type` - [in] The esp_http_client auth type

- **Return**
  - ESP_OK
  - ESP_ERR_INVALID_ARG

**int esp_http_client_get_errno (esp_http_client_handle_t client)

Get HTTP client session errno.

- **Parameters**
  - `client` - [in] The esp_http_client handle

- **Return**
  - (-1) if invalid argument
  - errno

**esp_err_t esp_http_client_set_method (esp_http_client_handle_t client, esp_http_client_method_t method)

Set http request method.
### Chapter 2. API 参考

**参数**
- `client` [in] The esp_http_client handle
- `method` [in] The method

**返回**
- ESP_OK
- ESP_ERR_INVALID_ARG

```c
esp_err_t esp_http_client_set_timeout_ms (esp_http_client_handle_t client, int timeout_ms)
```
Set http request timeout.

**参数**
- `client` [in] The esp_http_client handle
- `timeout_ms` [in] The timeout value

**返回**
- ESP_OK
- ESP_ERR_INVALID_ARG

```c
esp_err_t esp_http_client_delete_header (esp_http_client_handle_t client, const char* key)
```
Delete http request header.

**参数**
- `client` [in] The esp_http_client handle
- `key` [in] The key

**返回**
- ESP_OK
- ESP_FAIL

```c
int esp_http_client_open (esp_http_client_handle_t client, int write_len)
```
This function will be open the connection, write all header strings and return.

**参数**
- `client` [in] The esp_http_client handle
- `write_len` [in] HTTP Content length need to write to the server

**返回**
- ESP_OK
- ESP_FAIL

```c
int esp_http_client_write (esp_http_client_handle_t client, const char* buffer, int len)
```
This function will write data to the HTTP connection previously opened by esp_http_client_open()

**参数**
- `client` [in] The esp_http_client handle
- `buffer` The buffer
- `len` [in] This value must not be larger than the write_len parameter provided to esp_http_client_open()

**返回**
- (-1) if any errors
- Length of data written

```c
int64_t esp_http_client_fetch_headers (esp_http_client_handle_t client)
```
This function need to call after esp_http_client_open, it will read from http stream, process all receive headers.

**参数**
- `client` [in] The esp_http_client handle

**返回**
- (0) if stream doesn’t contain content-length header, or chunked encoding (checked by esp_http_client_is_chunked_response)
- (-1: ESP_FAIL) if any errors
- (-ESP_ERR_HTTP_EAGAIN = -0x7007) if call is timed-out before any data was ready
- Download data length defined by content-length header

```c
bool esp_http_client_is_chunked_response (esp_http_client_handle_t client)
```
Check response data is chunked.
### Chapter 2. API 参考

#### 参数 client - [in] The esp_http_client handle

#### 返回 true or false

```c
int esp_http_client_read (esp_http_client_handle_t client, char *buffer, int len)
```

Read data from http stream.

---

**备注:** (-ESP_ERR_HTTP_EAGAIN = -0x7007) is returned when call is timed-out before any data was ready

---

#### 参数

- client - [in] The esp_http_client handle
- buffer - The buffer
- len - [in] The length

#### 返回

- (-1) if any errors
- Length of data was read

```c
int esp_http_client_get_status_code (esp_http_client_handle_t client)
```

Get http response status code, the valid value if this function invoke after `esp_http_client_perform`

**参数**

- client - [in] The esp_http_client handle

**返回** Status code

```c
int64_t esp_http_client_get_content_length (esp_http_client_handle_t client)
```

Get http response content length (from header Content-Length) the valid value if this function invoke after `esp_http_client_perform`

**参数**

- client - [in] The esp_http_client handle

**返回**

- (-1) Chunked transfer
- Content-Length value as bytes

**esp_err_t**

```c
esp_http_client_close (esp_http_client_handle_t client)
```

Close http connection, still kept all http request resources.

**参数**

- client - [in] The esp_http_client handle

**返回**

- ESP_OK
- ESP_FAIL

**esp_err_t**

```c
esp_http_client_cleanup (esp_http_client_handle_t client)
```

This function must be the last function to call for an session. It is the opposite of the `esp_http_client_init` function and must be called with the same handle as input that a `esp_http_client_init` call returned. This might close all connections this handle has used and possibly has kept open until now. Don’t call this function if you intend to transfer more files, re-using handles is a key to good performance with esp_http_client.

**参数**

- client - [in] The esp_http_client handle

**返回**

- ESP_OK
- ESP_FAIL

**esp_http_client_transport_t**

```c
esp_http_client_get_transport_type (esp_http_client_handle_t client)
```

Get transport type.

**参数**

- client - [in] The esp_http_client handle

**返回**

- HTTP_TRANSPORT_UNKNOWN
- HTTP_TRANSPORT_OVER_TCP
- HTTP_TRANSPORT_OVER_SSL
**esp_err_t esp_http_client_set_redirection (esp_http_client_handle_t client)**

Set redirection URL. When received the 30x code from the server, the client stores the redirect URL provided by the server. This function will set the current URL to redirect to enable client to execute the redirection request.

参数 client - [in] The esp_http_client handle

返回
  • ESP_OK
  • ESP_FAIL

**void esp_http_client_add_auth (esp_http_client_handle_t client)**

On receiving HTTP Status code 401, this API can be invoked to add authorization information.

备注：There is a possibility of receiving body message with redirection status codes, thus make sure to flush off body data after calling this API.

参数 client - [in] The esp_http_client handle

**bool esp_http_client_is_complete_data_received (esp_http_client_handle_t client)**

Checks if entire data in the response has been read without any error.

参数 client - [in] The esp_http_client handle

返回
  • true
  • false

**int esp_http_client_read_response (esp_http_client_handle_t client, char *buffer, int len)**

Helper API to read larger data chunks. This is a helper API which internally calls esp_http_client_read multiple times till the end of data is reached or till the buffer gets full.

参数
  • client - [in] The esp_http_client handle
  • buffer - The buffer
  • len - [in] The buffer length

返回
  • Length of data was read

**esp_err_t esp_http_client_flush_response (esp_http_client_handle_t client, int *len)**

Process all remaining response data. This uses an internal buffer to repeatedly receive, parse, and discard response data until complete data is processed. As no additional user-supplied buffer is required, this may be preferable to esp_http_client_read_response in situations where the content of the response may be ignored.

参数
  • client - [in] The esp_http_client handle
  • len - Length of data discarded

返回
  • ESP_OK If successful, len will have discarded length
  • ESP_FAIL If failed to read response
  • ESP_ERR_INVALID_ARG If the client is NULL

**esp_err_t esp_http_client_get_url (esp_http_client_handle_t client, char *url, const int len)**

Get URL from client.

参数
  • client - [in] The esp_http_client handle
  • url - [inout] The buffer to store URL
  • len - [in] The buffer length

返回
  • ESP_OK
### Chapter 2. API 参考

**esp_err_t**  
`esp_http_client_get_chunk_length (esp_http_client_handle_t client, int *len)`  
Get Chunk-Length from client.

**参数**
- `client` - [in] The esp_http_client handle
- `len` - [out] Variable to store length

**返回**
- ESP_OK If successful, len will have length of current chunk
- ESP_FAIL If the server is not a chunked server
- ESP_ERR_INVALID_ARG If the client or len are NULL

**Structures**

`struct esp_http_client_event`

HTTP Client events data.

**Public Members**

`esp_http_client_event_id_t`  
`event_id`

event_id, to know the cause of the event

`esp_http_client_handle_t`  
`client`

esp_http_client_handle_t context

`void *data`

data of the event

`int data_len`

data length of data

`void *user_data`

user_data context, from `esp_http_client_config_t` user_data

`char *header_key`

For HTTP_EVENT_ON_HEADER event_id, it’s store current http header key

`char *header_value`

For HTTP_EVENT_ON_HEADER event_id, it’s store current http header value

`struct esp_http_client_config_t`

HTTP configuration.

**Public Members**

`const char *url`

HTTP URL, the information on the URL is most important, it overrides the other fields below, if any

`const char *host`

Domain or IP as string
int port
    Port to connect, default depend on esp_http_client_transport_t (80 or 443)

const char *username
    Using for Http authentication

const char *password
    Using for Http authentication

* esp_http_client_auth_type_t auth_type
    Http authentication type, see esp_http_client_auth_type_t

const char *path
    HTTP Path, if not set, default is /

const char *query
    HTTP query

const char *cert_pem
    SSL server certification, PEM format as string, if the client requires to verify server

size_t cert_len
    Length of the buffer pointed to by cert_pem. May be 0 for null-terminated pem

const char *client_cert_pem
    SSL client certification, PEM format as string, if the server requires to verify client

size_t client_cert_len
    Length of the buffer pointed to by client_cert_pem. May be 0 for null-terminated pem

const char *client_key_pem
    SSL client key, PEM format as string, if the server requires to verify client

size_t client_key_len
    Length of the buffer pointed to by client_key_pem. May be 0 for null-terminated pem

const char *client_key_password
    Client key decryption password string

size_t client_key_password_len
    String length of the password pointed to by client_key_password

const char *user_agent
    The User Agent string to send with HTTP requests

* esp_http_client_method_t method
    HTTP Method
int **timeout_ms**
Network timeout in milliseconds

bool **disable_auto_redirect**
Disable HTTP automatic redirects

int **max_redirection_count**
Max number of redirections on receiving HTTP redirect status code, using default value if zero

int **max_authorization_retries**
Max connection retries on receiving HTTP unauthorized status code, using default value if zero. Disables authorization retry if -1

**http_event_handle_cb**
HTTP Event Handle

**esp_http_client_transport_t**
HTTP transport type, see **esp_http_client_transport_t**

int **buffer_size**
HTTP receive buffer size

int **buffer_size_tx**
HTTP transmit buffer size

void * **user_data**
HTTP user_data context

bool **is_async**
Set asynchronous mode, only supported with HTTPS for now

bool **use_global_ca_store**
Use a global ca_store for all the connections in which this bool is set.

bool **skip_cert_common_name_check**
Skip any validation of server certificate CN field

const char * **common_name**
Pointer to the string containing server certificate common name. If non-NULL, server certificate CN must match this name. If NULL, server certificate CN must match hostname.

**esp_err_t** (*crt_bundle_attach*)(void *conf)
Function pointer to esp_crt_bundle_attach. Enables the use of certification bundle for server verification, must be enabled in menuconfig

bool **keep_alive_enable**
Enable keep-alive timeout
int `keep_alive_idle`
   Keep-alive idle time. Default is 5 (second)

int `keep_alive_interval`
   Keep-alive interval time. Default is 5 (second)

int `keep_alive_count`
   Keep-alive packet retry send count. Default is 3 counts

struct ifreq *`if_name`
   The name of interface for data to go through. Use the default interface without setting

**Macros**

`DEFAULT_HTTP_BUF_SIZE`

`ESP_ERR_HTTP_BASE`
   Starting number of HTTP error codes

`ESP_ERR_HTTP_MAX_REDIRECT`
   The error exceeds the number of HTTP redirects

`ESP_ERR_HTTP_CONNECT`
   Error open the HTTP connection

`ESP_ERR_HTTP_WRITE_DATA`
   Error write HTTP data

`ESP_ERR_HTTP_FETC/Header`
   Error read HTTP header from server

`ESP_ERR_HTTP_INVALID_TRANSPORT`
   There are no transport support for the input scheme

`ESP_ERR_HTTP_CONNECTING`
   HTTP connection hasn’t been established yet

`ESP_ERR_HTTP_EAGAIN`
   Mapping of errno EAGAIN to esp_err_t

`ESP_ERR_HTTP_CONNECTION_CLOSED`
   Read FIN from peer and the connection closed

**Type Definitions**

typedef struct esp_http_client *`esp_http_client_handle_t`

typedef struct `esp_http_client_event *esp_http_client_event_handle_t`
typedef struct esp_http_client_event esp_http_client_event_t
    HTTP Client events data.

typedef esp_err_t (*http_event_handle_cb)(esp_http_client_event_t *evt)

**Enumerations**

enum esp_http_client_event_id_t
    HTTP Client events id.

*Values:*

- **HTTP_EVENT_ERROR**
  This event occurs when there are any errors during execution

- **HTTP_EVENT_ON_CONNECTED**
  Once the HTTP has been connected to the server, no data exchange has been performed

- **HTTP_EVENT_HEADERS_SENT**
  After sending all the headers to the server

- **HTTP_EVENT_HEADER_SENT**
  This header has been kept for backward compatibility and will be deprecated in future versions esp-idf

- **HTTP_EVENT_ON_HEADER**
  Occurs when receiving each header sent from the server

- **HTTP_EVENT_ON_DATA**
  Occurs when receiving data from the server, possibly multiple portions of the packet

- **HTTP_EVENT_ON_FINISH**
  Occurs when finish a HTTP session

- **HTTP_EVENT_DISCONNECTED**
  The connection has been disconnected

- **HTTP_EVENT_REDIRECT**
  Intercepting HTTP redirects to handle them manually

enum esp_http_client_transport_t
    HTTP Client transport.

*Values:*

- **HTTP_TRANSPORT_UNKNOWN**
  Unknown

- **HTTP_TRANSPORT_OVER_TCP**
  Transport over tcp
enumerator HTTP_TRANSPORT_OVER_SSL
Transport over ssl

enum esp_http_client_method_t
HTTP method.

Values:

enumerator HTTP_METHOD_GET
HTTP GET Method

enumerator HTTP_METHOD_POST
HTTP POST Method

enumerator HTTP_METHOD_PUT
HTTP PUT Method

enumerator HTTP_METHOD_PATCH
HTTP PATCH Method

enumerator HTTP_METHOD_DELETE
HTTP DELETE Method

enumerator HTTP_METHOD_HEAD
HTTP HEAD Method

enumerator HTTP_METHOD_NOTIFY
HTTP NOTIFY Method

enumerator HTTP_METHOD_SUBSCRIBE
HTTP SUBSCRIBE Method

enumerator HTTP_METHOD_UNSUBSCRIBE
HTTP UNSUBSCRIBE Method

enumerator HTTP_METHOD_OPTIONS
HTTP OPTIONS Method

enumerator HTTP_METHOD_COPY
HTTP COPY Method

enumerator HTTP_METHOD_MOVE
HTTP MOVE Method

enumerator HTTP_METHOD_LOCK
HTTP LOCK Method

enumerator HTTP_METHOD_UNLOCK
HTTP UNLOCK Method
Chapter 2. API

enumerator HTTP_METHOD_PROPFIND
   HTTP PROPFIND Method

enumerator HTTP_METHOD_PROPPATCH
   HTTP PROPPATCH Method

enumerator HTTP_METHOD_MKCOL
   HTTP MKCOL Method

enumerator HTTP_METHOD_MAX

enum esp_http_client_auth_type_t
   HTTP Authentication type.
   Values:

   enumerator HTTP_AUTH_TYPE_NONE
      No authentication

   enumerator HTTP_AUTH_TYPE_BASIC
      HTTP Basic authentication

   enumerator HTTP_AUTH_TYPE_DIGEST
      HTTP Digest authentication

enum HttpStatus_Code
   Enum for the HTTP status codes.
   Values:

   enumerator HttpStatus_Ok

   enumerator HttpStatus_MultipleChoices

   enumerator HttpStatus_MovedPermanently

   enumerator HttpStatus_Found

   enumerator HttpStatus_SeeOther

   enumerator HttpStatus_TemporaryRedirect

   enumerator HttpStatus_PermanentRedirect

   enumerator HttpStatus_BadRequest

   enumerator HttpStatus_Unauthorized
enumerator HttpStatus_Forbidden

descriptor HttpStatus_NotFound

descriptor HttpStatus_InternalError

2.2.6 ESP Local Control

Overview

ESP Local Control (esp_local_ctrl) component in ESP-IDF provides capability to control an ESP device over Wi-Fi + HTTPS or BLE. It provides access to application defined properties that are available for reading / writing via a set of configurable handlers.

Initialization of the esp_local_ctrl service over BLE transport is performed as follows:

```c
esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_BLE,
    .transport_config = {
        .ble = & (protocomm_ble_config_t) {
            .device_name = SERVICE_NAME,
            .service_uuid = {
                /* LSB <------------------------
                   * ---------------------------------------> MSB */
                0x21,
                0xd5,
                0x3b,
                0x8d,
                0xbd,
                0x75,
                0x68,
                0x8a,
                0xb4,
                0x42,
                0xe8,
                0x31,
                0x4a,
                0x1e,
                0x98,
                0x3d
            }
        },
        .proto_sec = {
            .version = PROTOCOM_SEC0,
            .custom_handle = NULL,
            .sec_params = NULL
        },
        .handlers = {
            /* User defined handler functions */
            .get_prop_values = get_property_values,
            .set_prop_values = set_property_values,
            .usr_ctx = NULL,
            .usr_ctx_free_fn = NULL
        },
        /* Maximum number of properties that may be set */
        .max_properties = 10
    },
    /* Start esp_local_ctrl service */
    ESP_ERROR_CHECK(esp_local_ctrl_start(&config));
};
```

Similarly for HTTPS transport:

```c
/* Set the configuration */
httpd_ssl_config_t https_conf = HTTPD_SSL_CONFIG_DEFAULT();

/* Load server certificate */
extern const unsigned char servercert_start[] asm("_binary_servercert_pem_
  \_start");
extern const unsigned char servercert_end[] asm("_binary_servercert_pem_
  \_end");
https_conf.servercert = servercert_start;
```

(continued)
You may set security for transport in ESP local control using following options:

1. `PROTOCOM_SEC2`: specifies that SRP6a based key exchange and end to end encryption based on AES-GCM is used. This is the most preferred option as it adds a robust security with Augmented PAKE protocol i.e. SRP6a.

2. `PROTOCOM_SEC1`: specifies that Curve25519 based key exchange and end to end encryption based on AES-CTR is used.

3. `PROTOCOM_SEC0`: specifies that data will be exchanged as a plain text (no security).

4. `PROTOCOM_SEC_CUSTOM`: you can define your own security requirement. Please note that you will also have to provide `custom_handle` of type `protocom_security_t *` in this context.

备注：The respective security schemes need to be enabled through the project configuration menu. Please refer to the Enabling protocom security version section in Protocol Communication for more details.

Creating a property

Now that we know how to start the `esp_local_ctrl` service, let’s add a property to it. Each property must have a unique name (string), a type (e.g. enum), flags (bit fields) and size.

The size is to be kept 0, if we want our property value to be of variable length (e.g. if its a string or bytestream). For fixed length property value data-types, like int, float, etc., setting the size field to the right value, helps `esp_local_ctrl` to perform internal checks on arguments received with write requests.

The interpretation of type and flags fields is totally up to the application, hence they may be used as enumerations, bitfields, or even simple integers. One way is to use type values to classify properties, while flags to specify characteristics.
of a property.

Here is an example property which is to function as a timestamp. It is assumed that the application defines `TYPE_TIMESTAMP` and `READONLY`, which are used for setting the `type` and `flags` fields here.

```c
/* Create a timestamp property */
esp_local_ctrl_prop_t timestamp = {
    .name = "timestamp",
    .type = TYPE_TIMESTAMP,
    .size = sizeof(int32_t),
    .flags = READONLY,
    .ctx = func_get_time,
    .ctx_free_fn = NULL
};
/* Now register the property */
esp_local_ctrl_add_property(&timestamp);
```

Also notice that there is a `ctx` field, which is set to point to some custom `func_get_time()`. This can be used inside the property get / set handlers to retrieve timestamp.

Here is an example of `get_prop_values()` handler, which is used for retrieving the timestamp.

```c
static esp_err_t get_property_values(size_t props_count, const esp_local_ctrl_prop_t *props, esp_local_ctrl_prop_val_t *prop_values, void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        ESP_LOGI(TAG, "Reading %s", props[i].name);
        if (props[i].type == TYPE_TIMESTAMP) {
            /* Obtain the timer function from ctx */
            int32_t (*func_get_time)(void) = props[i].ctx;
            /* Use static variable for saving the value.
             * This is essential because the value has to be
             * valid even after this function returns.
             * Alternative is to use dynamic allocation
             * and set the free_fn field */
            static int32_t ts = func_get_time();
            prop_values[i].data = &ts;
        }
    }
    return ESP_OK;
}
```

Here is an example of `set_prop_values()` handler. Notice how we restrict from writing to read-only properties.

```c
static esp_err_t set_property_values(size_t props_count, const esp_local_ctrl_prop_t *props, const esp_local_ctrl_prop_val_t *prop_values, void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        if (props[i].flags & READONLY) {
            ESP_LOGE(TAG, "Cannot write to read-only property %s", props[i].name);
            return ESP_ERR_INVALID_ARG;
        } else {
            ESP_LOGI(TAG, "Setting %s", props[i].name);
        }
    }
}
```

(下页继续)
Chapter 2. API 参考

/* For keeping it simple, lets only log the incoming data */
ESP_LOG_BUFFER_HEX_LEVEL(TAG, prop_values[i].data,
    prop_values[i].size, ESP_LOG_INFO);
}
}
return ESP_OK;
}

For complete example see protocols/esp_local_ctrl

Client Side Implementation

The client side implementation will have establish a protocomm session with the device first, over the supported mode of transport, and then send and receive protobuf messages understood by the esp_local_ctrl service. The service will translate these messages into requests and then call the appropriate handlers (set / get). Then, the generated response for each handler is again packed into a protobuf message and transmitted back to the client.

See below the various protobuf messages understood by the esp_local_ctrl service:

1. get_prop_count: This should simply return the total number of properties supported by the service
2. get_prop_values: This accepts an array of indices and should return the information (name, type, flags) and values of the properties corresponding to those indices
3. set_prop_values: This accepts an array of indices and an array of new values, which are used for setting the values of the properties corresponding to the indices

Note that indices may or may not be the same for a property, across multiple sessions. Therefore, the client must only use the names of the properties to uniquely identify them. So, every time a new session is established, the client should first call get_prop_count and then get_prop_values, hence form an index to name mapping for all properties. Now when calling set_prop_values for a set of properties, it must first convert the names to indexes, using the created mapping. As emphasized earlier, the client must refresh the index to name mapping every time a new session is established with the same device.

The various protocomm endpoints provided by esp_local_ctrl are listed below:

<table>
<thead>
<tr>
<th>Endpoint Name (BLE + GATT Server)</th>
<th>URI (HTTPS Server + mDNS)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>esp_local_ctrl/version</td>
<td>https://&lt;mdns-hostname&gt;.local/esp_local_ctrl/version</td>
<td>Endpoint used for retrieving version string</td>
</tr>
<tr>
<td>esp_local_ctrl/control</td>
<td>https://&lt;mdns-hostname&gt;.local/esp_local_ctrl/control</td>
<td>Endpoint used for sending / receiving control messages</td>
</tr>
</tbody>
</table>

API Reference

Header File

- components/esp_local_ctrl/include/esp_local_ctrl.h

Functions

const esp_local_ctrl_transport_t *esp_local_ctrl_get_transport_ble (void)

Function for obtaining BLE transport mode.

const esp_local_ctrl_transport_t *esp_local_ctrl_get_transport_httpd (void)

Function for obtaining HTTPD transport mode.
**esp_err_t esp_local_ctrl_start** (const esp_local_ctrl_config_t *config)

Start local control service.

参数 `config` [in] Pointer to configuration structure

返回
- ESP_OK : Success
- ESP_FAIL : Failure

**esp_err_t esp_local_ctrl_stop** (void)

Stop local control service.

**esp_err_t esp_local_ctrl_add_property** (const esp_local_ctrl_prop_t *prop)

Add a new property.

This adds a new property and allocates internal resources for it. The total number of properties that could be added is limited by configuration option `max_properties`

参数 `prop` [in] Property description structure

返回
- ESP_OK : Success
- ESP_FAIL : Failure

**esp_err_t esp_local_ctrl_remove_property** (const char *name)

Remove a property.

This finds a property by name, and releases the internal resources which are associated with it.

参数 `name` [in] Name of the property to remove

返回
- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Failure

**const esp_local_ctrl_prop_t *esp_local_ctrl_get_property** (const char *name)

Get property description structure by name.

This API may be used to get a property’s context structure `esp_local_ctrl_prop_t` when its name is known

参数 `name` [in] Name of the property to find

返回
- Pointer to property
- NULL if not found

**esp_err_t esp_local_ctrl_set_handler** (const char *ep_name, prootcomm_reg_handler_t handler, void *user_ctx)

Register protocomm handler for a custom endpoint.

This API can be called by the application to register a protocomm handler for an endpoint after the local control service has started.

备注: In case of BLE transport the names and uuids of all custom endpoints must be provided beforehand as a part of the `protocomm_ble_config_t` structure set in `esp_local_ctrl_config_t`, and passed to `esp_local_ctrl_start()`.

参数
- `ep_name` -[in] Name of the endpoint
- `handler` -[in] Endpoint handler function
- `user_ctx` -[in] User data

返回
- ESP_OK : Success
- ESP_FAIL : Failure
Unions

union esp_local_ctrl_transport_config_t
    #include <esp_local_ctrl.h> Transport mode (BLE / HTTPD) configuration.

Public Members

esp_local_ctrl_transport_config_ble_t *ble
    This is same as protocomm_ble_config_t. See protocomm_ble.h for available configuration parameters.

esp_local_ctrl_transport_config_httpd_t *httpd
    This is same as httpd_ssl_config_t. See esp_https_server.h for available configuration parameters.

Structures

struct esp_local_ctrl_prop
    Property description data structure, which is to be populated and passed to the esp_local_ctrl_add_property() function.

Once a property is added, its structure is available for read-only access inside get_prop_values() and set_prop_values() handlers.

Public Members

char *name
    Unique name of property

uint32_t type
    Type of property. This may be set to application defined enums

size_t size
    Size of the property value, which:
    • if zero, the property can have values of variable size
    • if non-zero, the property can have values of fixed size only, therefore, checks are performed internally by esp_local_ctrl when setting the value of such a property

uint32_t flags
    Flags set for this property. This could be a bit field. A flag may indicate property behavior, e.g. read-only / constant

void *ctx
    Pointer to some context data relevant for this property. This will be available for use inside the get_prop_values and set_prop_values handlers as a part of this property structure. When set, this is valid throughout the lifetime of a property, till either the property is removed or the esp_local_ctrl service is stopped.

void (*ctx_free_fn)(void *ctx)
    Function used by esp_local_ctrl to internally free the property context when esp_local_ctrl_remove_property() or esp_local_ctrl_stop() is called.
struct `esp_local_ctrl_prop_val`  
Property value data structure. This gets passed to the `get_prop_values()` and `set_prop_values()` handlers for the purpose of retrieving or setting the present value of a property.

**Public Members**

- `void *data`
  Pointer to memory holding property value
- `size_t size`
  Size of property value
- `void (*free_fn)(void *data)`  
  This may be set by the application in `get_prop_values()` handler to tell `esp_local_ctrl` to call this function on the data pointer above, for freeing its resources after sending the `get_prop_values` response.

struct `esp_local_ctrl_handlers`  
Handlers for receiving and responding to local control commands for getting and setting properties.

**Public Members**

- `esp_err_t (*get_prop_values)(size_t props_count, const `esp_local_ctrl_prop_t` props[], `esp_local_ctrl_prop_val_t` prop_values[], void *usr_ctx)`  
  Handler function to be implemented for retrieving current values of properties.

<table>
<thead>
<tr>
<th>Param props_count [in]</th>
<th>Total elements in the props array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param props [in]</td>
<td>Array of properties, the current values for which have been requested by the client</td>
</tr>
<tr>
<td>Param prop_values [out]</td>
<td>Array of empty property values, the elements of which need to be populated with the current values of those properties specified by props argument</td>
</tr>
<tr>
<td>Param usr_ctx [in]</td>
<td>This provides value of theusr_ctx field of esp_local_ctrl_handlers_t structure</td>
</tr>
<tr>
<td>Return</td>
<td>Returning different error codes will convey the corresponding protocol level errors to the client:</td>
</tr>
<tr>
<td></td>
<td>• ESP_OK : Success</td>
</tr>
<tr>
<td></td>
<td>• ESP_ERR_INVALID_ARG : InvalidArgument</td>
</tr>
<tr>
<td></td>
<td>• ESP_ERR_INVALID_STATE : InvalidProto</td>
</tr>
<tr>
<td></td>
<td>• All other error codes : InternalError</td>
</tr>
</tbody>
</table>

- `esp_err_t (*set_prop_values)(size_t props_count, const `esp_local_ctrl_prop_t` props[], const `esp_local_ctrl_prop_val_t` prop_values[], void *usr_ctx)`  
  Handler function to be implemented for changing values of properties.

|备注：| If any of the properties have variable sizes, the size field of the corresponding element in prop_values must be checked explicitly before making any assumptions on the size.
**Param props_count** [in] Total elements in the props array

**Param props** [in] Array of properties, the values for which the client requests to change

**Param prop_values** [in] Array of property values, the elements of which need to be used for updating those properties specified by props argument

**Param usr_ctx** [in] This provides value of the **usr_ctx** field of `esp_local_ctrl_handlers_t` structure

**Return** Returning different error codes will convey the corresponding protocol level errors to the client:
- ESP_OK: Success
- ESP_ERR_INVALID_ARG: InvalidArgument
- ESP_ERR_INVALID_STATE: InvalidProto
- All other error codes: InternalError

```c
void *usr_ctx
```

Context pointer to be passed to above handler functions upon invocation. This is different from the property level context, as this is valid throughout the lifetime of the `esp_local_ctrl` service, and freed only when the service is stopped.

```c
void (*usr_ctx_free_fn)(void *usr_ctx)
```

Pointer to function which will be internally invoked on `usr_ctx` for freeing the context resources when `esp_local_ctrl_stop()` is called.

```c
struct esp_local_ctrl_proto_sec_cfg
```

Protocom security configs

**Public Members**

```c
esp_local_ctrl_proto_sec_t version
```

This sets protocom security version, sec0/sec1 or custom If custom, user must provide handle via `proto_sec_custom_handle` below

```c
void *custom_handle
```

Custom security handle if security is set custom via `proto_sec` above This handle must follow `protocomm_security_t` signature

```c
const void *pop
```

Proof of possession to be used for local control. Could be NULL.

```c
const void *sec_params
```

Pointer to security params (NULL if not needed). This is not needed for protocomm security 0 This pointer should hold the struct of type `esp_local_ctrl_security1_params_t` for protocomm security 1 and `esp_local_ctrl_security2_params_t` for protocomm security 2 respectively. Could be NULL.

```c
struct esp_local_ctrl_config
```

Configuration structure to pass to `esp_local_ctrl_start()`

**Public Members**

```c
const esp_local_ctrl_transport_t *transport
```

Transport layer over which service will be provided
**esp_local_ctrl_transport_config_t** transport_config
Transport layer over which service will be provided

**esp_local_ctrl_proto_sec_cfg_t** proto_sec
Security version and POP

**esp_local_ctrl_handlers_t** handlers
Register handlers for responding to get/set requests on properties

**size_t** max_properties
This limits the number of properties that are available at a time

**Macros**

**ESP_LOCAL_CTRL_TRANSPORT_BLE**

**ESP_LOCAL_CTRL_TRANSPORT_HTTPD**

**Type Definitions**

typedef struct **esp_local_ctrl_prop** esp_local_ctrl_prop_t
Property description data structure, which is to be populated and passed to the esp_local_ctrl_add_property() function.

Once a property is added, its structure is available for read-only access inside get_prop_values() and set_prop_values() handlers.

typedef struct **esp_local_ctrl_prop_val** esp_local_ctrl_prop_val_t
Property value data structure. This gets passed to the get_prop_values() and set_prop_values() handlers for the purpose of retrieving or setting the present value of a property.

typedef struct **esp_local_ctrl_handlers** esp_local_ctrl_handlers_t
Handlers for receiving and responding to local control commands for getting and setting properties.

typedef struct esp_local_ctrl_transport esp_local_ctrl_transport_t
Transport mode (BLE / HTTPD) over which the service will be provided.

This is forward declaration of a private structure, implemented internally by esp_local_ctrl.t

**protocomm_ble_config esp_local_ctrl_transport_config_ble_t**
Configuration for transport mode BLE.

This is a forward declaration for protocomm_ble_config_t. To use this, application must set CONFIG_BT_BLUEDROID_ENABLED and include protocomm_ble.h.

**httpd_ssl_config esp_local_ctrl_transport_config_httpd_t**
Configuration for transport mode HTTPD.

This is a forward declaration for httpd_ssl_config_t. To use this, application must set CONFIG_ESP_HTTPS_SERVER_ENABLE and include esp_https_server.h

**typedef enum esp_local_ctrl_proto_sec esp_local_ctrl_proto_sec_t**
Security types for esp_local_control.
typedef protocomm_security1_params_t esp_local_ctrl_security1_params_t
typedef protocomm_security2_params_t esp_local_ctrl_security2_params_t
typedef struct esp_local_ctrl_proto_sec_cfg esp_local_ctrl_proto_sec_cfg_t
    Protocom security configs
typedef struct esp_local_ctrl_config esp_local_ctrl_config_t
    Configuration structure to pass to esp_local_ctrl_start()

Enumerations
enum esp_local_ctrl_proto_sec
    Security types for esp_local_control.
    Values:
    enumerator PROTOCOM_SEC0
    enumerator PROTOCOM_SEC1
    enumerator PROTOCOM_SEC2
    enumerator PROTOCOM_SEC_CUSTOM

2.2.7 ESP Serial Slave Link

Overview
Espressif provides several chips that can work as slaves. These slave devices rely on some common buses, and have their own communication protocols over those buses. The esp_serial_slave_link component is designed for the master to communicate with ESP slave devices through those protocols over the bus drivers.
After an esp_serial_slave_link device is initialized properly, the application can use it to communicate with the ESP slave devices conveniently.

Espressif Device protocols
For more details about Espressif device protocols, see the following documents.

Communication with ESP SDIO Slave
This document describes the process of initialization of an ESP SDIO Slave device and then provides details on the ESP SDIO Slave protocol - a non-standard protocol that allows an SDIO Host to communicate with an ESP SDIO slave.
The ESP SDIO Slave protocol was created to implement the communication between SDIO host and slave, because the SDIO specification only shows how to access the custom region of a card (by sending CMD52 and CMD53 to Functions 1-7) without any details regarding the underlying hardware implementation.
**SDIO Slave Capabilities of Espressif chips**  The services provided by the SDIO Slave peripheral of the ESP32 chip are listed in the table below:

<table>
<thead>
<tr>
<th>Services</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDIO slave</td>
<td>Y</td>
</tr>
<tr>
<td>Tohost intr</td>
<td>8</td>
</tr>
<tr>
<td>Frhost intr</td>
<td>8</td>
</tr>
<tr>
<td>TX DMA</td>
<td>Y</td>
</tr>
<tr>
<td>RX DMA</td>
<td>Y</td>
</tr>
<tr>
<td>Shared registers</td>
<td>56*</td>
</tr>
</tbody>
</table>

- * Not including the interrupt registers

**ESP SDIO Slave Initialization**  The host should initialize the ESP32 SDIO slave according to the standard SDIO initialization process (Section 3.1.2 of *SDIO Simplified Specification*). In this specification as well as below, the SDIO slave is called an (SD)IO card. Here is a brief example of an ESP SDIO Slave initialization process:

1. **SDIO reset** CMD52 (Write 0x6=0x8)
2. **SD reset** CMD0
3. **Check whether IO card (optional)** CMD8
4. **Send SDIO op cond and wait for card ready** CMD5 arg = 0x00000000  
   CMD5 arg = 0x00FF8000 (according to the response above, poll until ready)  
   **Example**: Arg of R4 after first CMD5 (arg=0x00000000) is 0xxxFFFFF00.  
   Keep sending CMD5 with arg=0x00FFFFF00 until the R4 shows card ready (arg bit 31=1).
5. **Set address** CMD3
6. **Select card** CMD7 (arg address according to CMD3 response)  
   **Example**: Arg of R6 after CMD3 is 0x0001xxxx.  
   Arg of CMD7 should be 0x00010000.
7. **Select 4-bit mode (optional)** CMD52 (Write 0x07=0x02)
8. **Enable func1** CMD52 (Write 0x02=0x02)
9. **Enable SDIO interrupt (required if interrupt line (DAT1) is used)** CMD52 (Write 0x04=0x03)
10. **Set Func0 blocksize (optional, default value is 512 (0x200))** CMD52/53 (Read 0x10~0x11)  
    CMD52/53 (Write 0x10=0x00)  
    CMD52/53 (Write 0x11=0x02)  
    CMD52/53 (Read 0x10~0x11, read to check the final value)
11. **Set Func1 blocksize (optional, default value is 512 (0x200))** CMD52/53 (Read 0x110~0x111)  
    CMD52/53 (Write 0x110=0x00)  
    CMD52/53 (Write 0x111=0x02)  
    CMD52/53 (Read 0x110~0x111, read to check the final value)

**ESP SDIO Slave Protocol**  The ESP SDIO Slave protocol is based on the SDIO Specification’s I/O Read/Write commands, i.e., CMD52 and CMD53. The protocol offers the following services:

- Sending FIFO and receiving FIFO
- 52 8-bit R/W registers shared by host and slave (For details, see *ESP32 Technical Reference Manual > SDIO Slave Controller > Register Summary > SDIO SLC Host registers [PDF]*)
- 16 general purpose interrupt sources, 8 from host to slave and 8 from slave to host

To begin communication, the host needs to enable the I/O Function 1 in the slave and access its registers as described below.

Check the code example `peripherals/sdio`.

The **ESP Serial Slave Link** component implements the logic of this protocol for ESP32 SDIO Host when communicating with an ESP32 SDIO slave.

**Slave register table**
Chapter 2. API

32-bit

- 0x044 (TOKEN_RDATA): in which bit 27-16 holds the number of the receiving buffer.
- 0x058 (INT_ST): holds the interrupt source bits from slave to host.
- 0x060 (PKT_LEN): holds the accumulated data length (in bytes) already read by host plus the data copied to the buffer but yet to be read.
- 0x0D4 (INT_CLR): write 1 to clear interrupt bits corresponding to INT_ST.
- 0x0DC (INT_ENA): mask bits for interrupts from slave to host.

8-bit

Shared general purpose registers:

- 0x06C-0x077: R/W registers 0-11 shared by slave and host.
- 0x07A-0x07B: R/W registers 14-15 shared by slave and host.
- 0x07E-0x07F: R/W registers 18-19 shared by slave and host.
- 0x088-0x08B: R/W registers 24-27 shared by slave and host.
- 0x09C-0x0BB: R/W registers 32-63 shared by slave and host.

Interrupt Registers: - 0x08D (SLAVE_INT): bits for host to interrupt slave. auto clear.

FIFO (sending and receiving) 0x090 - 0x1F7FF are reserved for FIFOs.

The address of CMD53 is related to the length requested to read from or write to the slave in a single transfer, as demonstrated by the equation below:

\[
\text{requested length} = 0x1F800 - \text{address}
\]

The slave will respond with data that has a length equal to the length field of CMD53. In cases where the data is longer than the requested length, the data will be zero filled (when sending) or discarded (when receiving). This includes both the block and the byte mode of CMD53.

**Interrupts**

SDIO interrupts are “level sensitive”. For host interrupts, the slave sends an interrupt by pulling the DAT1 line down at a proper time. The host detects when the interrupt line is pulled down and reads the INT_ST register to determine the source of the interrupt. After that, the host can clear the interrupt bits by writing the INT_CLR register and process the interrupt. The host can also mask unneeded sources by clearing the bits in the INT_ENA register corresponding to the sources. If all the sources are cleared (or masked), the DAT1 line goes inactive.

On ESP32, the corresponding host_int bits are: bit 0 to bit 7.

For slave interrupts, the host sends a transfer to write the SLAVE_INT register. Once a bit is set to 1, the slave hardware and the driver will detect it and inform the application.

**Receiving FIFO**

To write to the slave’s receiving FIFO, the host should complete the following steps:

1. **Read the TOKEN1 field (bits 27-16) of the register TOKEN_RDATA (0x044)**. The buffer number remaining is TOKEN1 minus the number of buffers used by host.
2. **Make sure the buffer number is sufficient** (buffer_size x buffer_num is greater than the data to write, buffer_size is pre-defined between the host and the slave before the communication starts). Otherwise, keep returning to Step 1 until the buffer size is sufficient.
3. **Write to the FIFO address with CMD53.** Note that the requested length should not exceed the length calculated at Step 2, and the FIFO address is related to requested length.

4. **Calculate used buffers.** Note that a partially used buffer at the tail is counted as used.

**Sending FIFO** To read the slave’s sending FIFO, the host should complete the following steps:

1. **Wait for the interrupt line to become active** (optional, low by default).
2. **Read (poll) the interrupt bits in the INT_ST register** to monitor if new packets exist.
3. **If new packets are ready, read the PKT_LEN register.** Before reading the packets, determine the length of data to be read. As the host keeps the length of data already read from the slave, subtract this value from PKT_LEN, the result will be the maximum length of data available for reading. If no data has been added to the sending FIFO yet, wait and poll until the slave is ready and update PKT_LEN.
4. **Read from the FIFO using CMD53.** Note that the requested length should not be greater than calculated at Step 3, and the FIFO address is related to requested length.
5. **Update the read length.**

警告：The driver for ESP32 hasn’t been developed yet.

**ESP SPI Slave HD (Half Duplex) Mode Protocol**

<table>
<thead>
<tr>
<th>SPI Slave Capabilities of Espressif chips</th>
<th>ESP32</th>
<th>ESP32-S2</th>
<th>ESP32-C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI Slave HD</td>
<td>N</td>
<td>Y (v2)</td>
<td>Y (v2)</td>
</tr>
<tr>
<td>To host intr</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Fp host intr</td>
<td>2 *</td>
<td>2 *</td>
<td></td>
</tr>
<tr>
<td>TX DMA</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>RX DMA</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Shared registers</td>
<td>72</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

**Introduction** In the half duplex mode, the master has to use the protocol defined by the slave to communicate with the slave. Each transaction may consist of the following phases (list by the order they should exist):

- **Command:** 8-bit, master to slave
  
  This phase determines the rest phases of the transactions. See Supported Commands.

- **Address:** 8-bit, master to slave, optional
  
  For some commands (WRBUF, RDBUF), this phase specifies the address of the shared buffer to write/to read from. For other commands with this phase, they are meaningless but still have to exist in the transaction.

- **Dummy:** 8-bit, floating, optional
  
  This phase is the turnaround time between the master and the slave on the bus, and also provides enough time for the slave to prepare the data to send to the master.

- **Data:** variable length, the direction is also determined by the command.
  
  This may be a data OUT phase, in which the direction is slave to master, or a data IN phase, in which the direction is master to slave.

The direction means which side (master or slave) controls the MOSI, MISO, WP, and HD pins.

**Data IO Modes** In some IO modes, more data wires can be used to send the data. As a result, the SPI clock cycles required for the same amount of data will be less than in the 1-bit mode. For example, in QIO mode, address and data (IN and OUT) should be sent on all 4 data wires (MOSI, MISO, WP, and HD). Here are the modes supported by the ESP32-S2 SPI slave and the wire number used in corresponding modes.
Normally, which mode is used is determined by the command sent by the master (See Supported Commands), except the QPI mode.

**QPI Mode**  The QPI mode is a special state of the SPI Slave. The master can send the ENQPI command to put the slave into the QPI mode state. In the QPI mode, the command is also sent in 4-bit, thus it’s not compatible with the normal modes. The master should only send QPI commands when the slave is in QPI mode. To exit from the QPI mode, master can send the EXQPI command.

### Supported Commands

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Command</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRBUF</td>
<td>Write buffer</td>
<td>0x01</td>
<td>Buf addr</td>
<td>master to slave, no longer than buffer size</td>
</tr>
<tr>
<td>RDBUF</td>
<td>Read buffer</td>
<td>0x02</td>
<td>Buf addr</td>
<td>slave to master, no longer than buffer size</td>
</tr>
<tr>
<td>WRDMA</td>
<td>Write DMA</td>
<td>0x03</td>
<td>8 bits</td>
<td>master to slave, no longer than length provided by slave</td>
</tr>
<tr>
<td>RDDMA</td>
<td>Read DMA</td>
<td>0x04</td>
<td>8 bits</td>
<td>slave to master, no longer than length provided by slave</td>
</tr>
<tr>
<td>SEG_DONE</td>
<td>Segments done</td>
<td>0x05</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>ENQPI</td>
<td>Enter QPI mode</td>
<td>0x06</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>WR_DONE</td>
<td>Write segments done</td>
<td>0x07</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>CMD8</td>
<td>Interrupt</td>
<td>0x08</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>CMD9</td>
<td>Interrupt</td>
<td>0x09</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>CMDA</td>
<td>Interrupt</td>
<td>0x0A</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>EXQPI</td>
<td>Exit QPI mode</td>
<td>0xDD</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

Moreover, WRBUF, RDBUF, WRDMA, RDDMA commands have their 2-bit and 4-bit version. To do transactions in 2-bit or 4-bit mode, send the original command ORed by the corresponding command mask below. For example, command 0xA1 means WRBUF in QIO mode.
### Segment Transaction Mode
Segment transaction mode is the only mode supported by the SPI Slave HD driver for now. In this mode, for a transaction the slave load onto the DMA, the master is allowed to read or write in segments. This way the master doesn’t have to prepare a large buffer as the size of data provided by the slave. After the master finishes reading/writing a buffer, it has to send the corresponding termination command to the slave as a synchronization signal. The slave driver will update new data (if exist) onto the DMA upon seeing the termination command.

The termination command is WR_DONE (0x07) for the WRDMA and CMD8 (0x08) for the RDDMA.

Here’s an example for the flow the master read data from the slave DMA:

1. The slave loads 4092 bytes of data onto the RDDMA
2. The master do seven RDDMA transactions, each of them is 512 bytes long, and reads the first 3584 bytes from the slave
3. The master do the last RDDMA transaction of 512 bytes (equal, longer, or shorter than the total length loaded by the slave are all allowed). The first 508 bytes are valid data from the slave, while the last 4 bytes are meaningless bytes.
4. The master sends CMD8 to the slave
5. The slave loads another 4092 bytes of data onto the RDDMA
6. The master can start new reading transactions after it sends the CMD8

### Terminology
- **ESSL**: Abbreviation for ESP Serial Slave Link, the component described by this document.
- **Master**: The device running the esp_serial_slave_link component.
- **ESSL device**: A virtual device on the master associated with an ESP slave device. The device context has the knowledge of the slave protocol above the bus, relying on some bus drivers to communicate with the slave.
- **ESSL device handle**: A handle to ESSL device context containing the configuration, status and data required by the ESSL component. The context stores the driver configurations, communication state, data shared by master and slave, etc.
  
  The context should be initialized before it is used, and get deinitialized if not used any more. The master application operates on the ESSL device through this handle.
- **ESP slave**: The slave device connected to the bus, which ESSL component is designed to communicate with.
- **Bus**: The bus over which the master and the slave communicate with each other.
- **Slave protocol**: The special communication protocol specified by Espressif HW/SW over the bus.
- **TX buffer num**: A counter, which is on the slave and can be read by the master, indicates the accumulated buffer numbers that the slave has loaded to the hardware to receive data from the master.
- **RX data size**: A counter, which is on the slave and can be read by the master, indicates the accumulated data size that the slave has loaded to the hardware to send to the master.

### Services provided by ESP slave

There are some common services provided by the Espressif slaves:

1. **Tohost Interrupts**: The slave can inform the master about certain events by the interrupt line. (optional)
2. **Frohost Interrupts**: The master can inform the slave about certain events.
3. **Tx FIFO (master to slave)**: The slave can send data in stream to the master. The SDIO slave can also indicate it has new data to send to master by the interrupt line.
The slave updates the TX buffer num to inform the master how much data it can receive, and the master then read the TX buffer num, and take off the used buffer number to know how many buffers are remaining.
4. Rx FIFO (slave to master): the slave can receive data from the master in units of receiving buffers.
   The slave updates the RX data size to inform the master how much data it has prepared to send, and then the master read the data size, and take off the data length it has already received to know how many data is remaining.
5. Shared registers: the master can read some part of the registers on the slave, and also write these registers to let the slave read.

The services provided by the slave depends on the slave’s model. See *SDIO Slave Capabilities of Expressif chips* and *SPI Slave Capabilities of Expressif chips* for more details.

**Initialization of ESP Serial Slave Link**

**ESP SDIO Slave**  
The ESP SDIO slave link (ESSL SDIO) devices relies on the sdmmcc component. It includes the usage of communicating with ESP SDIO Slave device via SDSPI feature. The ESSL device should be initialized as below:

1. Initialize a sdmmcc card (see doc: Document of SDMMC driver <api-reference/storage/sdmmc>) structure.
2. Call `sdmmc_card_init()` to initialize the card.
3. Initialize the ESSL device with `essl_sdio_config_t`. The card member should be the `sdmmc_card_t` got in step 2, and the recv_buffer_size member should be filled correctly according to pre-negotiated value.
4. Call `essl_init()` to do initialization of the SDIO part.
5. Call `essl_wait_for_ready()` to wait for the slave to be ready.

**ESP SPI Slave**  
备注： If you are communicating with the ESP SDIO Slave device through SPI interface, you should use the *SDIO interface* instead.

Hasn’t been supported yet.

**APIs**

After the initialization process above is performed, you can call the APIs below to make use of the services provided by the slave:

**Tohost Interrupts (optional)**

1. Call `essl_get_intr_ena()` to know which events will trigger the interrupts to the master.
2. Call `essl_set_intr_ena()` to set the events that will trigger interrupts to the master.
3. Call `essl_wait_int()` to wait until interrupt from the slave, or timeout.
4. When interrupt is triggered, call `essl_get_intr()` to know which events are active, and call `essl_clear_intr()` to clear them.

**Frhost Interrupts**

1. Call `essl_send_slave_intr()` to trigger general purpose interrupt of the slave.

**TX FIFO**

1. Call `essl_get_tx_buffer_num()` to know how many buffers the slave has prepared to receive data from the master. This is optional. The master will poll `tx_buffer_num` when it try to send packets to the slave, until the slave has enough buffer or timeout.
2. Call `essl_send_packet()` to send data to the slave.
RX FIFO

1. Call \texttt{essl\_get\_rx\_data\_size()} to know how many data the slave has prepared to send to the master. This is optional. When the master tries to receive data from the slave, it will update the \textit{rx\_data\_size} for once, if the current \textit{rx\_data\_size} is shorter than the buffer size the master prepared to receive. And it may poll the \textit{rx\_data\_size} if the \textit{rx\_dat\_size} keeps 0, until timeout.

2. Call \texttt{essl\_get\_packet()} to receive data from the slave.

Reset counters (Optional) Call \texttt{essl\_reset\_cnt()} to reset the internal counter if you find the slave has reset its counter.

Application Example

The example below shows how ESP32 SDIO host and slave communicate with each other. The host use the ESSL SDIO peripherals/sdio.

Please refer to the specific example README.md for details.

API Reference

Header File

- \texttt{components/driver/test\_apps/components/esp\_serial\_slave\_link/include/esp\_serial\_slave\_link/essl.h}

Functions

\texttt{esp\_err\_t essl\_init (essl\_handle\_t handle, uint32\_t wait\_ms)}

Initialize the slave.

- \texttt{handle} - Handle of an ESSL device.
- \texttt{wait\_ms} - Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: If success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- Other value returned from lower layer \texttt{init}.

\texttt{esp\_err\_t essl\_wait\_for\_ready (essl\_handle\_t handle, uint32\_t wait\_ms)}

Wait for interrupt of an ESSL slave device.

- \texttt{handle} - Handle of an ESSL device.
- \texttt{wait\_ms} - Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: If success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- One of the error codes from SDMMC host controller

\texttt{esp\_err\_t essl\_get\_tx\_buffer\_num (essl\_handle\_t handle, uint32\_t *out\_tx\_num, uint32\_t wait\_ms)}

Get buffer num for the host to send data to the slave. The buffers are size of \textit{buffer\_size}.

- \texttt{handle} - Handle of a ESSL device.
- \texttt{out\_tx\_num} - Output of buffer num that host can send data to ESSL slave.
- \texttt{wait\_ms} - Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode
• One of the error codes from SDMMC/SPI host controller

```c
esp_err_t essl_get_rx_data_size(essl_handle_t handle, uint32_t*out_rx_size, uint32_t wait_ms)
```

Get the size, in bytes, of the data that the ESSL slave is ready to send.

参数
- **handle** - Handle of an ESSL device.
- **out_rx_size** - Output of data size to read from slave, in bytes
- **wait_ms** - Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回
- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode

```c
esp_err_t essl_reset_cnt (essl_handle_t handle)
```

Reset the counters of this component. Usually you don’t need to do this unless you know the slave is reset.

参数 **handle** - Handle of an ESSL device.

返回
- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode

```c
esp_err_t essl_send_packet (essl_handle_t handle, const void* start, size_t length, uint32_t wait_ms)
```

Send a packet to the ESSL Slave. The Slave receives the packet into buffers whose size is buffer_size (configured during initialization).

参数
- **handle** - Handle of an ESSL device.
- **start** - Start address of the packet to send
- **length** - Length of data to send, if the packet is over-size, the it will be divided into blocks and hold into different buffers automatically.
- **wait_ms** - Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回
- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid argument, handle is not init or other argument is not valid.
- ESP_ERR_TIMEOUT: No buffer to use, or error from SDMMC host controller.
- ESP_ERR_NOT_FOUND: Slave is not ready for receiving.
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode

```c
esp_err_t essl_get_packet (essl_handle_t handle, void*out_data, size_t size, size_t*out_length, uint32_t wait_ms)
```

Get a packet from ESSL slave.

参数
- **handle** - Handle of an ESSL device.
- **out_data** - [out] Data output address
- **size** - The size of the output buffer, if the buffer is smaller than the size of data to receive from slave, the driver returns ESP_ERR_NOT_FINISHED
- **out_length** - [out] Output of length the data actually received from slave.
- **wait_ms** - Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回
- ESP_OK: Success: All the data has been read from the slave.
- ESP_ERR_INVALID_ARG: Invalid argument, The handle is not initialized or the other arguments are invalid.
- ESP_ERR_NOT_FINISHED: Read was successful, but there is still data remaining.
- ESP_ERR_NOT_FOUND: Slave is not ready to send data.
- ESP_ERR_NOT_SUPPORTED: This API is not supported in this mode

One of the error codes from SDMMC/SPI host controller.
```c
esp_err_t essl_write_reg(essl_handle_t handle, uint8_t addr, uint8_t value, uint8_t* value_o, uint32_t wait_ms)
```

Write general purpose R/W registers (8-bit) of ESSL slave.

<table>
<thead>
<tr>
<th>参数</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle – Handle of an ESSL device.</td>
<td></td>
</tr>
<tr>
<td>addr – Address of register to write. For SDIO, valid address: 0-59. For SPI, see essl_spi.h</td>
<td>sdio 28-31 are reserved, the lower API helps to skip.</td>
</tr>
<tr>
<td>value – Value to write to the register.</td>
<td></td>
</tr>
<tr>
<td>value_o – Output of the returned written value.</td>
<td></td>
</tr>
<tr>
<td>wait_ms – Millisecond to wait before timeout, will not wait at all if set to 0-9.</td>
<td></td>
</tr>
</tbody>
</table>

返回
- ESP_OK Success
- One of the error codes from SDMMC/SPI host controller

```c
esp_err_t essl_read_reg(essl_handle_t handle, uint8_t add, uint8_t* value_o, uint32_t wait_ms)
```

Read general purpose R/W registers (8-bit) of ESSL slave.

<table>
<thead>
<tr>
<th>参数</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle – Handle of an ESSL device.</td>
<td></td>
</tr>
<tr>
<td>add – Address of register to read. For SDIO, Valid address: 0-27, 32-63 (28-31 reserved, return interrupt bits on read). For SPI, see essl_spi.h</td>
<td></td>
</tr>
<tr>
<td>value_o – Output value read from the register.</td>
<td></td>
</tr>
<tr>
<td>wait_ms – Millisecond to wait before timeout, will not wait at all if set to 0-9.</td>
<td></td>
</tr>
</tbody>
</table>

返回
- ESP_OK Success
- One of the error codes from SDMMC/SPI host controller

```c
esp_err_t essl_wait_int(essl_handle_t handle, uint32_t wait_ms)
```

wait for an interrupt of the slave

<table>
<thead>
<tr>
<th>参数</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle – Handle of an ESSL device.</td>
<td></td>
</tr>
<tr>
<td>wait_ms – Millisecond to wait before timeout, will not wait at all if set to 0-9.</td>
<td></td>
</tr>
</tbody>
</table>

返回
- ESP_OK: If interrupt is triggered.
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- ESP_ERR_TIMEOUT: No interrupts before timeout.

```c
esp_err_t essl_clear_intr(essl_handle_t handle, uint32_t intr_mask, uint32_t wait_ms)
```

Clear interrupt bits of ESSL slave. All the bits set in the mask will be cleared, while other bits will stay the same.

<table>
<thead>
<tr>
<th>参数</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle – Handle of an ESSL device.</td>
<td></td>
</tr>
<tr>
<td>intr_mask – Mask of interrupt bits to clear.</td>
<td></td>
</tr>
<tr>
<td>wait_ms – Millisecond to wait before timeout, will not wait at all if set to 0-9.</td>
<td></td>
</tr>
</tbody>
</table>

返回
- ESP_OK: Success
- ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
- One of the error codes from SDMMC host controller

```c
esp_err_t essl_get_intr(essl_handle_t handle, uint32_t *intr_raw, uint32_t *intr_st, uint32_t wait_ms)
```

Get interrupt bits of ESSL slave.

<table>
<thead>
<tr>
<th>参数</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle – Handle of an ESSL device.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2. API

• **intr_raw** – Output of the raw interrupt bits. Set to NULL if only masked bits are read.
• **intr_st** – Output of the masked interrupt bits. Set to NULL if only raw bits are read.
• **wait_ms** – Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

• ESP_OK: Success
• ESP_INVALID_ARG: If both **intr_raw** and **intr_st** are NULL.
• ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
• One of the error codes from SDMMC host controller

**esp_err_t essl_set_intr_ena (essl_handle_t handle, uint32_t ena_mask, uint32_t wait_ms)**

Set interrupt enable bits of ESSL slave. The slave only sends interrupt on the line when there is a bit both the raw status and the enable are set.

参数

• **handle** – Handle of an ESSL device.
• **ena_mask** – Mask of the interrupt bits to enable.
• **wait_ms** – Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

• ESP_OK: Success
• ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
• One of the error codes from SDMMC host controller

**esp_err_t essl_get_intr_ena (essl_handle_t handle, uint32_t* ena_mask_o, uint32_t wait_ms)**

Get interrupt enable bits of ESSL slave.

参数

• **handle** – Handle of an ESSL device.
• **ena_mask_o** – Output of interrupt bit enable mask.
• **wait_ms** – Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

• ESP_OK: Success
• One of the error codes from SDMMC host controller

**esp_err_t essl_send_slave_intr (essl_handle_t handle, uint32_t intr_mask, uint32_t wait_ms)**

Send interrupts to slave. Each bit of the interrupt will be triggered.

参数

• **handle** – Handle of an ESSL device.
• **intr_mask** – Mask of interrupt bits to send to slave.
• **wait_ms** – Millisecond to wait before timeout, will not wait at all if set to 0-9.

返回

• ESP_OK: Success
• ESP_ERR_NOT_SUPPORTED: Current device does not support this function.
• One of the error codes from SDMMC host controller

Type Definitions

typedef struct essl_dev_t *essl_handle_t

Handle of an ESSL device.

Header File

• components/driver/test_apps/components/esp_serial_slave_link/include/esp_serial_slave_link/essl_sdio.h

Functions

**esp_err_t essl_sdio_init_dev (essl_handle_t *out_handle, const essl_sdio_config_t *config)**

Initialize the ESSL SDIO device and get its handle.

参数
Chapter 2. API 参考

- **out_handle** - Output of the handle.
- **config** - Configuration for the ESSL SDIO device.

**返回**
- **ESP_OK**: on success
- **ESP_ERR_NO_MEM**: memory exhausted.

```c
esp_err_t essl_sdio_deinit_dev (essl_handle_t handle)
```

Deinitialize and free the space used by the ESSL SDIO device.

**参数**
- **handle** - Handle of the ESSL SDIO device to deinit.

**返回**
- **ESP_OK**: on success
- **ESP_ERR_INVALID_ARG**: wrong handle passed

**Structures**

```c
struct essl_sdio_config_t
```

Configuration for the ESSL SDIO device.

**Public Members**

```c
sdmmc_card_t *card
```

The initialized sdmmc card pointer of the slave.

```c
int recv_buffer_size
```

The pre-negotiated recv buffer size used by both the host and the slave.

**Header File**

- components/driver/test_apps/components/esp_serial_slave_link/include/esp_serial_slave_link/essl_spi.h

**Functions**

```c
esp_err_t essl_spi_init_dev (essl_handle_t *out_handle, const essl_spi_config_t *init_config)
```

Initialize the ESSL SPI device function list and get its handle.

**参数**
- **out_handle** - [out] Output of the handle
- **init_config** - Configuration for the ESSL SPI device

**返回**
- **ESP_OK**: On success
- **ESP_ERR_NO_MEM**: Memory exhausted
- **ESP_ERR_INVALID_STATE**: SPI driver is not initialized
- **ESP_ERR_INVALID_ARG**: Wrong register ID

```c
esp_err_t essl_spi_deinit_dev (essl_handle_t handle)
```

Deinitialize the ESSL SPI device and free the memory used by the device.

**参数**
- **handle** - Handle of the ESSL SPI device

**返回**
- **ESP_OK**: On success
- **ESP_ERR_INVALID_STATE**: ESSL SPI is not in use

```c
esp_err_t essl_spi_read_reg (void *arg, uint8_t addr, uint8_t *out_value, uint32_t wait_ms)
```

Read from the shared registers.
The registers for Master/Slave synchronization are reserved. Do not use them. (see `rx_sync_reg` in `essl_spi_config_t`)

**Parameters**

- `arg` - Context of the component. (Member `arg` from `essl_handle_t`)
- `addr` - Address of the shared registers. (Valid: 0 ~ SOC_SPI_MAXIMUM_BUFFER_SIZE, registers for M/S sync are reserved, see note1).
- `out_value` - [out] Read buffer for the shared registers.
- `wait_ms` - Time to wait before timeout (reserved for future use, user should set this to 0).

**Return**

- ESP_OK: success
- ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
- ESP_ERR_INVALID_ARG: The address argument is not valid. See note 1.
- or other return value from `spi_device_transmit`.

```c
esp_err_t essl_spi_get_packet(void *arg, void *out_data, size_t size, uint32_t wait_ms)
```

Get a packet from Slave.

**Parameters**

- `arg` - Context of the component. (Member `arg` from `essl_handle_t`)
- `out_data` - [out] Output data address
- `size` - The size of the output data.
- `wait_ms` - Time to wait before timeout (reserved for future use, user should set this to 0).

**Return**

- ESP_OK: On Success
- ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
- ESP_ERR_INVALID_ARG: The output data address is neither DMA capable nor 4 byte-aligned
- ESP_ERR_INVALID_SIZE: Master requires size bytes of data but Slave did not load enough bytes.

```c
esp_err_t essl_spi_write_reg(void *arg, uint8_t addr, uint8_t value, uint8_t *out_value, uint32_t wait_ms)
```

Write to the shared registers.

**Parameters**

- `arg` - Context of the component. (Member `arg` from `essl_handle_t`)
- `addr` - Address of the shared registers. (Valid: 0 ~ SOC_SPI_MAXIMUM_BUFFER_SIZE, registers for M/S sync are reserved, see note1)
- `value` - Buffer for data to send, should be align to 4.
- `out_value` - [out] Not supported, should be set to NULL.
- `wait_ms` - Time to wait before timeout (reserved for future use, user should set this to 0).

**Return**

- ESP_OK: success

**Note:** Feature of checking the actual written value (`out_value`) is not supported.

```c
esp_err_t essl_spi_write_reg(void *arg, uint8_t addr, uint8_t value, uint8_t *out_value, uint32_t wait_ms)
```
• ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
• ESP_ERR_INVALID_ARG: The address argument is not valid. See note 1.
• ESP_ERR_NOT_SUPPORTED: Should set out_value to NULL. See note 2.
• or other return value from: \cpp\func\spi_device_transmit.

\textbf{esp_err_t essl_spi_send_packet} (void *arg, const void *data, size_t size, uint32_t wait_ms)
Send a packet to Slave.

### 参数
- \textbf{arg} – Context of the component. (Member \texttt{arg} from \texttt{essl_handle_t})
- \textbf{data} – Address of the data to send
- \textbf{size} – Size of the data to send.
- \textbf{wait_ms} – Time to wait before timeout (reserved for future use, user should set this to 0).

### 返回
- ESP_OK: On success
- ESP_ERR_INVALID_STATE: ESSL SPI has not been initialized.
- ESP_ERR_INVALID_ARG: The data address is not DMA capable
- ESP_ERR_INVALID_SIZE: Master will send size bytes of data but Slave did not load enough RX buffer

\textbf{void essl_spi_reset_cnt} (void *arg)
Reset the counter in Master context.

### 备注:
Shall only be called if the slave has reset its counter. Else, Slave and Master would be desynchronized.

### 参数 \textbf{arg} – Context of the component. (Member \texttt{arg} from \texttt{essl_handle_t})

\textbf{esp_err_t essl_spi_rdbuf} (spi_device_handle_t spi, uint8_t *out_data, int addr, int len, uint32_t flags)
Read the shared buffer from the slave in ISR way.

### 备注:
The slave’s HW doesn’t guarantee the data in one SPI transaction is consistent. It sends data in unit of byte. In other words, if the slave SW attempts to update the shared register when a rdbuf SPI transaction is in-flight, the data got by the master will be the combination of bytes of different writes of slave SW.

### 备注:
\texttt{out_data} should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the \texttt{len} is shorter than a word.

### 参数
- \textbf{spi} – SPI device handle representing the slave
- \textbf{out_data} – [out] Buffer for read data, strongly suggested to be in the DRAM and aligned to 4
- \textbf{addr} – Address of the slave shared buffer
- \textbf{len} – Length to read
- \textbf{flags} – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

### 返回
- ESP_OK: on success
- or other return value from \cpp\func\spi_device_transmit.

\textbf{esp_err_t essl_spi_rdbuf_polling} (spi_device_handle_t spi, uint8_t *out_data, int addr, int len, uint32_t flags)
Read the shared buffer from the slave in polling way.
Chapter 2. API

备注: out_data should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the len is shorter than a word.

参数
- spi – SPI device handle representing the slave
- out_data – [out] Buffer for read data, strongly suggested to be in the DRAM and aligned to 4
- addr – Address of the slave shared buffer
- len – Length to read
- flags – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

返回
- ESP_OK: on success
- or other return value from \cpp{func}{spi_device_transmit}.

\cpp{func}{essl_spi_wrbuf}(spi_device_handle_t spi, const uint8_t* data, int addr, int len, uint32_t flags)

Write the shared buffer of the slave in ISR way.

备注: out_data should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the len is shorter than a word.

参数
- spi – SPI device handle representing the slave
- data – Buffer for data to send, strongly suggested to be in the DRAM
- addr – Address of the slave shared buffer
- len – Length to write
- flags – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

返回
- ESP_OK: success
- or other return value from \cpp{func}{spi_device_transmit}.

\cpp{func}{essl_spi_wrbuf_polling}(spi_device_handle_t spi, const uint8_t* data, int addr, int len, uint32_t flags)

Write the shared buffer of the slave in polling way.

备注: out_data should be prepared in words and in the DRAM. The buffer may be written in words by the DMA. When a byte is written, the remaining bytes in the same word will also be overwritten, even the len is shorter than a word.

参数
- spi – SPI device handle representing the slave
- data – Buffer for data to send, strongly suggested to be in the DRAM
- addr – Address of the slave shared buffer
- len – Length to write
- flags – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

返回
- ESP_OK: success
- or other return value from \cpp{func}{spi_device_polling_transmit}.

\cpp{func}{essl_spi_rddma}(spi_device_handle_t spi, uint8_t* out_data, int len, int seg_len, uint32_t flags)

Receive long buffer in segments from the slave through its DMA.
This function combines several \cpp{essl_spi_rddma_seg} and one \cpp{essl_spi_rddma_done} at the end. Used when the slave is working in segment mode.

**Parameters**
- **spi** – SPI device handle representing the slave
- **out_data** – [out] Buffer to hold the received data, strongly suggested to be in the DRAM and aligned to 4
- **len** – Total length of data to receive.
- **seg_len** – Length of each segment, which is not larger than the maximum transaction length allowed for the spi device. Suggested to be multiples of 4. When set < 0, means send all data in one segment (the rddma_done will still be sent.)
- **flags** – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

**Retuns**
- ESP_OK: success
- or other return value from \cpp{spi_device_transmit}.

\cpp{esp_err_t essl_spi_rddma_seg(spi_device_handle_t spi, uint8_t *out_data, int seg_len, uint32_t flags)}
Read one data segment from the slave through its DMA.

**备注:** To read long buffer, call \cpp{essl_spi_rddma} instead.

**Parameters**
- **spi** – SPI device handle representing the slave
- **out_data** – [out] Buffer to hold the received data, strongly suggested to be in the DRAM and aligned to 4
- **seg_len** – Length of this segment
- **flags** – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

**Returns**
- ESP_OK: success
- or other return value from \cpp{spi_device_transmit}.

\cpp{esp_err_t essl_spi_rddma_done(spi_device_handle_t spi, uint32_t flags)}
Send the rddma_done command to the slave. Upon receiving this command, the slave will stop sending the current buffer even there are data unsent, and maybe prepare the next buffer to send.

**备注:** This is required only when the slave is working in segment mode.

**Parameters**
- **spi** – SPI device handle representing the slave
- **flags** – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

**Returns**
- ESP_OK: success
- or other return value from \cpp{spi_device_transmit}.

\cpp{esp_err_t essl_spi_wrdma(spi_device_handle_t spi, const uint8_t *data, int len, int seg_len, uint32_t flags)}
Send long buffer in segments to the slave through its DMA.

**备注:** This function combines several \cpp{essl_spi_wrdma_seg} and one \cpp{essl_spi_wrdma_done} at the end. Used when the slave is working in segment mode.
• `spi` – SPI device handle representing the slave
• `data` – Buffer for data to send, strongly suggested to be in the DRAM
• `len` – Total length of data to send.
• `seg_len` – Length of each segment, which is not larger than the maximum transaction length allowed for the spi device. Suggested to be multiples of 4. When set < 0, means send all data in one segment (the `wrdma_done` will still be sent.)
• `flags` – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

### Return
• ESP_OK: success
• or other return value from `cpp:func:spi_device_transmit`.

```c
esp_err_t essl_spi_wrdma_seg(spi_device_handle_t spi, const uint8_t* data, int seg_len, uint32_t flags)
```
Send one data segment to the slave through its DMA.

**Note:** To send long buffer, call `cpp:func:essl_spi_wrdma` instead.

### Parameter
• `spi` – SPI device handle representing the slave
• `data` – Buffer for data to send, strongly suggested to be in the DRAM
• `seg_len` – Length of this segment
• `flags` – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

### Return
• ESP_OK: success
• or other return value from `cpp:func:spi_device_transmit`.

```c
esp_err_t essl_spi_wrdma_done(spi_device_handle_t spi, uint32_t flags)
```
Send the `wrdma_done` command to the slave. Upon receiving this command, the slave will stop receiving, process the received data, and maybe prepare the next buffer to receive.

**Note:** This is required only when the slave is working in segment mode.

### Parameter
• `spi` – SPI device handle representing the slave
• `flags` – SPI_TRANS_* flags to control the transaction mode of the transaction to send.

### Return
• ESP_OK: success
• or other return value from `cpp:func:spi_device_transmit`.

### Structures

```c
struct essl_spi_config_t
```
Configuration of ESSL SPI device.

### Public Members

```c
spi_device_handle_t *spi
```
Pointer to SPI device handle.

```c
uint32_t tx_buf_size
```
The pre-negotiated Master TX buffer size used by both the host and the slave.
uint8_t tx_sync_reg
The pre-negotiated register ID for Master-TX-SLAVE-RX synchronization. 1 word (4 Bytes) will be reserved for the synchronization.

uint8_t rx_sync_reg
The pre-negotiated register ID for Master-RX-Slave-TX synchronization. 1 word (4 Bytes) will be reserved for the synchronization.

2.2.8 ESP x509 Certificate Bundle

Overview

The ESP x509 Certificate Bundle API provides an easy way to include a bundle of custom x509 root certificates for TLS server verification.

The bundle is currently not available when using WolfSSL.

The bundle comes with the complete list of root certificates from Mozilla’s NSS root certificate store. Using the gen_crt_bundle.py python utility the certificates’ subject name and public key are stored in a file and embedded in the ESP32 binary.

When generating the bundle you may choose between:

- The full root certificate bundle from Mozilla, containing more than 130 certificates. The current bundle was updated Tue Oct 11 03:12:05 2022 GMT.
- A pre-selected filter list of the name of the most commonly used root certificates, reducing the amount of certificates to around 35 while still having around 90% coverage according to market share statistics.

In addition it is possible to specify a path to a certificate file or a directory containing certificates which then will be added to the generated bundle.

Trusting all root certificates means the list will have to be updated if any of the certificates are retracted. This includes removing them from cacrt_all.pem.

Configuration

Most configuration is done through menuconfig. CMake will generate the bundle according to the configuration and embed it.

- CONFIG_MBEDTLS_CERTIFICATE_BUNDLE: automatically build and attach the bundle.
- CONFIG_MBEDTLS_DEFAULT_CERTIFICATE_BUNDLE: decide which certificates to include from the complete root list.
- CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE_PATH: specify the path of any additional certificates to embed in the bundle.

To enable the bundle when using ESP-TLS simply pass the function pointer to the bundle attach function:

```c
esp_tls_cfg_t cfg = {
    .crt_bundle_attach = esp_crt_bundle_attach,
};
```

This is done to avoid embedding the certificate bundle unless activated by the user.

If using mbedTLS directly then the bundle may be activated by directly calling the attach function during the setup process.
Generating the List of Root Certificates

The list of root certificates comes from Mozilla’s NSS root certificate store, which can be found here. The list can be downloaded and created by running the script `mk-ca-bundle.pl` that is distributed as a part of curl. Another alternative would be to download the finished list directly from the curl website: CA certificates extracted from Mozilla.

The common certificates bundle were made by selecting the authorities with a market share of more than 1% from w3tech’s SSL Survey. These authorities were then used to pick the names of the certificates for the filter list, `cmn_crtAuthorities.csv`, from this list provided by Mozilla.

Updating the Certificate Bundle

The bundle is embedded into the app and can be updated along with the app by an OTA update. If you want to include a more up-to-date bundle than the bundle currently included in ESP-IDF, then the certificate list can be downloaded from Mozilla as described in Generating the List of Root Certificates.

Application Example

Simple HTTPS example that uses ESP-TLS to establish a secure socket connection using the certificate bundle with two custom certificates added for verification: `protocols/https_x509_bundle`.

HTTPS example that uses ESP-TLS and the default bundle: `protocols/https_request`.

HTTPS example that uses mbedTLS and the default bundle: `protocols/https_mbedtls`.

API Reference

Header File

- `components/mbedtls/esp_crt_bundle/include/esp_crt_bundle.h`

Functions

```c
esp_err_t esp_crt_bundle_attach(void *conf)
```

Attach and enable use of a bundle for certificate verification.

- `conf` - [in] The config struct for the SSL connection.
- ESP_OK if adding certificates was successful.
- Other if an error occurred or an action must be taken by the calling process.

```c
void esp_crt_bundle_detach(mbedtls_ssl_config *conf)
```

Disable and dealloc the certification bundle.

- `conf` - [in] The config struct for the SSL connection.
- Removes the certificate verification callback and deallocates used resources.
esp_err_t esp_crt_bundle_set (const uint8_t *x509_bundle, size_t bundle_size)

Set the default certificate bundle used for verification.

Overrides the default certificate bundle only in case of successful initialization. In most use cases the bundle should be set through menuconfig. The bundle needs to be sorted by subject name since binary search is used to find certificates.

参数

返回
- ESP_OK if adding certificates was successful.
- Other if an error occured or an action must be taken by the calling process.

2.2.9 HTTP 服务器

概述

HTTP Server 组件提供了在 ESP32 上运行轻量级 Web 服务器的功能，下面介绍使用 HTTP Server 组件 API 的详细步骤:

- httpd_start(): 创建 HTTP 服务器的实例，根据具体的配置为其分配内存和资源，并返回该服务器实例的句柄。服务器使用了两个套接字，一个用于监听 HTTP 流量 (TCP 类型)，另一个用来处理控制信号（UDP 类型），它们在服务器的任务循环中轮流使用。通过向 httpd_start() 传递 httpd_config_t 结构体，可以在创建服务器实例时配置任务的优先级和堆栈的大小。TCP 流量被解析为 HTTP 请求，根据请求的 URI 来调用用户注册的处理程序。在处理程序中需要发送回 HTTP 响应数据包。
- httpd_stop(): 根据传入的句柄停止服务器，并释放相关的内存和资源。这是一个阻塞函数，首先给服务器任务发送停止信号，然后等待其终止。期间服务器任务会关闭所有已打开的连接，删除已注册的 URI 处理程序，并将所有会话的上下文数据重置为空。
- httpd_register_uri_handler(): 通过传入 httpd_uri_t 结构体类型的对象来注册 URI 处理程序。该结构体包含如下成员: uri 名字，method 类型（比如 HTTPD_GET/HTTPD_POST/HTTPD_PUT 等等），esp_err_t *handler (httpd_req_t *req) 类型的函数指针，指向用户上下文数据的 user_ctx 指针。

应用示例

```c
/* URI 处理函数，在客户端发起 GET /uri 请求时被调用 */
esp_err_t get_handler (httpd_req_t *req)
{
    /* 发送回简单的响应数据包 */
    const char[] resp = "URI GET Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

/* URI 处理函数，在客户端发起 POST/uri 请求时被调用 */
esp_err_t post_handler (httpd_req_t *req)
{
    /* 定义 HTTP POST 请求数据的目标缓冲区 */
    char content[100];
    httpd_req_recv(req, HTTPD_RECV_REPLACE);
    /* 任意二进制数据（需要类型转换）
    * 对于字符串数据，null 终止符会被省略，
    * content_len 会给出字符串的长度 */
    // 假定这里的内容是字符串
    char content[100];
    // 此处用于处理字符串数据
```
Chapter 2. API

```c
/* 如果内容长度大于缓冲区则截断 */
size_t recv_size = MIN(req->content_len, sizeof(content));

int ret = httpd_req_recv(req, content, recv_size);
if (ret <= 0) { /* 返回 0 表示连接已关闭 */
  /* 检查是否超时 */
  if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
    /* 如果是超时，可以调用 httpd_req_recv() 重试 */
    /* 简单起见，这里我们直接 */
    /* 响应 HTTP 408（请求超时）错误给客户端 */
    httpd_resp_send_408(req);
  }
  /* 如果发生了错误，返回 ESP_FAIL 可以确保 */
  /* 底层套接字被关闭 */
  return ESP_FAIL;
}

/* 发送简单的响应数据包 */
const char[] resp = "URI POST Response";
httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
return ESP_OK;

/* GET /uri 的 URI 处理结构 */
httpd_uri_t uri_get = {
  .uri = "/uri",
  .method = HTTP_GET,
  .handler = get_handler,
  .user_ctx = NULL
};

/* POST/uri 的 URI 处理结构 */
httpd_uri_t uri_post = {
  .uri = "/uri",
  .method = HTTP_POST,
  .handler = post_handler,
  .user_ctx = NULL
};

/* 启动 Web 服务器的函数 */
httpd_handle_t start_webserver(void)
{
  /* 生成默认的配置参数 */
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();

  /* 置空 esp_http_server 的实例句柄 */
  httpd_handle_t server = NULL;

  /* 启动 httpd server */
  if (httpd_start(&server, &config) == ESP_OK) {
    /* 注册 URI 处理程序 */
    httpd_register_uri_handler(server, &uri_get);
    httpd_register_uri_handler(server, &uri_post);
  }
  /* 如果服务器启动失败，返回的句柄是 NULL */
  return server;
}

/* 停止 Web 服务器的函数 */
void stop_webserver(httpd_handle_t server)
```
简单 HTTP 服务器示例 请查看位于 protocols/http_server/simple 的 HTTP 服务器示例，该示例演示了如何处理任意内容长度的数据，读取请求头和 URL 查询参数，设置响应头。

HTTP 长连接

HTTP 服务器具有长连接的功能，允许重复使用同一个连接（会话）进行多次传输，同时保持会话的上下文数据。上下文数据可由处理程序动态分配，在这种情况下需要提前指定自定义的回调函数，以便在连接/会话被关闭时释放这部分内存资源。

长连接示例

```c
/* 自定义函数，用来释放上下文数据 */
void free_ctx_func(void *ctx)
{
    /* 也可以是 free 以外的代码逻辑 */
    free(ctx);
}

esp_err_t adder_post_handler(httpd_req_t *req)
{
    /* 若上下文中不存在会话，则新建一个 */
    if (!req->sess_ctx) {
        req->sess_ctx = malloc(sizeof(ANY_DATA_TYPE)); /*！< 指向上下文数据 */
        req->free_ctx = free_ctx_func;
        /*！< 释放上下文数据的函数 */
    }

    /* 访问上下文数据 */
    ANY_DATA_TYPE *ctx_data = (ANY_DATA_TYPE *)req->sess_ctx;

    /* 响应 */
    .................
    .................

    return ESP_OK;
}
```

详情请参考位于 protocols/http_server/persistent_sockets 的示例代码。

Websocket 服务器

HTTP 服务器组件提供 websocket 支持。可以在 menuconfig 中使用 CONFIG_HTTPD_WS_SUPPORT 选项启用 websocket 功能。有关如何使用 websocket 功能，请参阅 protocols/http_server/ws_echo_server 目录下的示例代码。

API 参考

Header File
• components/esp_http_server/include/esp_http_server.h

**Functions**

```c
esp_err_t httpd_register_uri_handler(httpd_handle_t handle, const httpd_uri_t *uri_handler)
```

Registers a URI handler.

Example usage:

```c
esp_err_t my_uri_handler(httpd_req_t *req)
{
    // Recv, Process and Send
    ....
    ....
    ....

    // Fail condition
    if (....) {
        // Return fail to close session
        return ESP_FAIL;
    }

    // On success
    return ESP_OK;
}

// URI handler structure
httpd_uri_t my_uri {
    .uri = "/my_uri/path/xyz",
    .method = HTTPD_GET,
    .handler = my_uri_handler,
    .user_ctx = NULL
};

// Register handler
if (httpd_register_uri_handler(server_handle, &my_uri) != ESP_OK) {
    // If failed to register handler
    ....
}
```

**Notes:** URI handlers can be registered in real time as long as the server handle is valid.
• **method** [in] HTTP method

返回
• ESP_OK: On successfully deregistering the handler
• ESP_ERR_INVALID_ARG: Null arguments
• ESP_ERR_NOT_FOUND: Handler with specified URI and method not found

**esp_err_t httpd_unregister_uri (httpd_handle_t handle, const char *uri)**

Unregister all URI handlers with the specified uri string.

参数
• **handle** [in] handle to HTTPD server instance
• **uri** [in] uri string specifying all handlers that need to be deregistered

返回
• ESP_OK: On successfully deregistering all such handlers
• ESP_ERR_INVALID_ARG: Null arguments
• ESP_ERR_NOT_FOUND: No handler registered with specified uri string

**esp_err_t httpd_sess_set_recv_override (httpd_handle_t hd, int sockfd, httpd_recv_func_t recv_func)**

Override web server’s receive function (by session FD)

This function overrides the web server’s receive function. This same function is used to read HTTP request packets.

备注: 此API意应在必要时从上下文中被调用，包括
• an http session API where sockfd is a valid parameter
• a URI handler where sockfd is obtained using httpd_req_to_sockfd()

参数
• **hd** [in] HTTPD instance handle
• **sockfd** [in] Session socket FD
• **recv_func** [in] The receive function to be set for this session

返回
• ESP_OK: On successfully registering override
• ESP_ERR_INVALID_ARG: Null arguments

**esp_err_t httpd_sess_set_send_override (httpd_handle_t hd, int sockfd, httpd_send_func_t send_func)**

Override web server’s send function (by session FD)

This function overrides the web server’s send function. This same function is used to send out any response to any HTTP request.

备注: 此API意应在必要时从上下文中被调用，包括
• an http session API where sockfd is a valid parameter
• a URI handler where sockfd is obtained using httpd_req_to_sockfd()

参数
• **hd** [in] HTTPD instance handle
• **sockfd** [in] Session socket FD
• **send_func** [in] The send function to be set for this session

返回
• ESP_OK: On successfully registering override
• ESP_ERR_INVALID_ARG: Null arguments

**esp_err_t httpd_sess_set_pending_override (httpd_handle_t hd, int sockfd, httpd_pending_func_t pending_func)**

参数
• **hd** [in] HTTPD instance handle
• **sockfd** [in] Session socket FD
• **send_func** [in] The send function to be set for this session

返回
• ESP_OK: On successfully registering override
• ESP_ERR_INVALID_ARG: Null arguments
Override web server’s pending function (by session FD)

This function overrides the web server’s pending function. This function is used to test for pending bytes in a socket.

备注：This API is supposed to be called either from the context of
• an http session APIs where sockfd is a valid parameter
• a URI handler where sockfd is obtained using httpd_req_to_sockfd()

参数
• hd  [in] HTTPD instance handle
• sockfd  [in] Session socket FD
• pending_func  [in] The receive function to be set for this session

返回
• ESP_OK : On successfully registering override
• ESP_ERR_INVALID_ARG : Null arguments

int httpd_req_to_sockfd (httpd_req_t *r)

Get the Socket Descriptor from the HTTP request.

This API will return the socket descriptor of the session for which URI handler was executed on reception of HTTP request. This is useful when user wants to call functions that require session socket fd, from within a URI handler, ie. : httpd_sess_get_ctx(), httpd_sess_trigger_close(), httpd_sess_update_lru_counter().

备注：This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.

参数 r  [in] The request whose socket descriptor should be found

返回
• Socket descriptor : The socket descriptor for this request
• -1 : Invalid/NULL request pointer

int httpd_req_recv (httpd_req_t *r, char *buf, size_t buf_len)

API to read content data from the HTTP request.

This API will read HTTP content data from the HTTP request into provided buffer. Use content_len provided in httpd_req_t structure to know the length of data to be fetched. If content_len is too large for the buffer then user may have to make multiple calls to this function, each time fetching ‘buf_len’ number of bytes, while the pointer to content data is incremented internally by the same number.

备注：
• This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
• If an error is returned, the URI handler must further return an error. This will ensure that the erroneous socket is closed and cleaned up by the web server.
• Presently Chunked Encoding is not supported

参数
• r  [in] The request being responded to
• buf  [in] Pointer to a buffer that the data will be read into
• buf_len  [in] Length of the buffer

返回
• Bytes : Number of bytes read into the buffer successfully
Chapter 2. API

- 0: Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID: Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT: Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL: Unrecoverable error while calling socket recv()

\[
\text{size_t httpd_req_get_hdr_value_len (httpd_req_t *r, const char *field)}
\]

Search for a field in request headers and return the string length of it’s value.

**备注:**
- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once httpd_resp_send() API is called all request headers are purged, so request headers need be copied into separate buffers if they are required later.

**参数**
- \( r \) [in] The request being responded to
- \( field \) [in] The header field to be searched in the request

返回
- Length: If field is found in the request URL
- Zero: Field not found / Invalid request / Null arguments

\[
\text{esp_err_t httpd_req_get_hdr_value_str (httpd_req_t *r, const char *field, char *val, size_t val_size)}
\]

Get the value string of a field from the request headers.

**备注:**
- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once httpd_resp_send() API is called all request headers are purged, so request headers need be copied into separate buffers if they are required later.
- If output size is greater than input, then the value is truncated, accompanied by truncation error as return value.
- Use httpd_req_get_hdr_value_len() to know the right buffer length

**参数**
- \( r \) [in] The request being responded to
- \( field \) [in] The field to be searched in the header
- \( val \) [out] Pointer to the buffer into which the value will be copied if the field is found
- \( val\_size \) [in] Size of the user buffer “val”

返回
- ESP_OK: Field found in the request header and value string copied
- ESP_ERR_NOT_FOUND: Key not found
- ESP_ERR_INVALID_ARG: Null arguments
- ESP_ERR_HTTPD_INVALID_REQ: Invalid HTTP request pointer
- ESP_ERR_HTTPD_RESULT_TRUNC: Value string truncated

\[
\text{size_t httpd_req_get_url_query_len (httpd_req_t *r)}
\]

Get Query string length from the request URL.

**备注:** This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid

**参数** \( r \) [in] The request being responded to
Chapter 2. API

**esp_err_t** httpd_req_get_url_query_str(httpd_req_t *r, char* buf, size_t buf_len)

Get Query string from the request URL.

**备注：**

- Presently, the user can fetch the full URL query string, but decoding will have to be performed by the user. Request headers can be read using httpd_req_get_hdr_value_str() to know the ‘Content-Type’ (eg. Content-Type: application/x-www-form-urlencoded) and then the appropriate decoding algorithm needs to be applied.
- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- If output size is greater than input, then the value is truncated, accompanied by truncation error as return value.
- Prior to calling this function, one can use httpd_req_get_url_query_len() to know the query string length beforehand and hence allocate the buffer of right size (usually query string length + 1 for null termination) for storing the query string.

**参数**

- **r** -[in] The request being responded to
- **buf** -[out] Pointer to the buffer into which the query string will be copied (if found)
- **buf_len** -[in] Length of output buffer

**返回**

- **ESP_OK** : Query is found in the request URL and copied to buffer
- **ESP_ERR_NOT_FOUND** : Query not found
- **ESP_ERR_INVALID_ARG** : Null arguments
- **ESP_ERR_HTTPD_INVALID_REQ** : Invalid HTTP request pointer
- **ESP_ERR_HTTPD_RESULT_TRUNC** : Query string truncated

**esp_err_t** httpd_query_key_value(const char*qry, const char*key, char*val, size_t val_size)

Helper function to get a URL query tag from a query string of the type param1=val1&param2=val2.

**备注：**

- The components of URL query string (keys and values) are not URL decoded. The user must check for ‘Content-Type’ field in the request headers and then depending upon the specified encoding (URL encoded or otherwise) apply the appropriate decoding algorithm.
- If actual value size is greater than val_size, then the value is truncated, accompanied by truncation error as return value.

**参数**

- **qry** -[in] Pointer to query string
- **key** -[in] The key to be searched in the query string
- **val** -[out] Pointer to the buffer into which the value will be copied if the key is found
- **val_size** -[in] Size of the user buffer “val”

**返回**

- **ESP_OK** : Key is found in the URL query string and copied to buffer
- **ESP_ERR_NOT_FOUND** : Key not found
- **ESP_ERR_INVALID_ARG** : Null arguments
- **ESP_ERR_HTTPDRESULT_TRUNC** : Value string truncated
**esp_err_t** httpd_req_get_cookie_val (httpd_req_t *req, const char *cookie_name, char *val, size_t *val_size)

Get the value string of a cookie value from the “Cookie” request headers by cookie name.

**参数**
- `req` - [in] Pointer to the HTTP request
- `cookie_name` - [in] The cookie name to be searched in the request
- `val` - [out] Pointer to the buffer into which the value of cookie will be copied if the cookie is found
- `val_size` - [inout] Pointer to size of the user buffer “val”. This variable will contain cookie length if ESP_OK is returned and required buffer length in case ESP_ERR_HTTPD_RESULT_TRUNC is returned.

**返回**
- ESP_OK: Key is found in the cookie string and copied to buffer
- ESP_ERR_NOT_FOUND: Key not found
- ESP_ERR_INVALID_ARG: Null arguments
- ESP_ERR_HTTPD_RESULT_TRUNC: Value string truncated
- ESP_ERR_NO_MEM: Memory allocation failure

**bool** httpd_uri_matchWildcard (const char *uri_template, const char *uri_to_match, size_t match_upto)

Test if a URI matches the given wildcard template.

Template may end with “?” to make the previous character optional (typically a slash), “*” for a wildcard match, and “?*” to make the previous character optional, and if present, allow anything to follow.

**Example:**
- * matches everything
- /foo/ matches /foo and /foo/
- /foo/* (sans the backslash) matches /foo/ and /foo/bar, but not /foo or /fo
- /foo/? or /foo/*? (sans the backslash) matches /foo/, /foo/bar, and also /foo, but not /foox or /fo

The special characters “?” and “*” anywhere else in the template will be taken literally.

**参数**
- `uri_template` - [in] URI template (pattern)
- `uri_to_match` - [in] URI to be matched
- `match_upto` - [in] how many characters of the URI buffer to test (there may be trailing query string etc.)

**返回**
- true if a match was found

**esp_err_t** httpd_resp_send (httpd_req_t *r, const char *buf, ssize_t buf_len)

API to send a complete HTTP response.

This API will send the data as an HTTP response to the request. This assumes that you have the entire response ready in a single buffer. If you wish to send response in incremental chunks use httpd_resp_send_chunk() instead.

If no status code and content-type were set, by default this will send 200 OK status code and content type as text/html. You may call the following functions before this API to configure the response headers:
- httpd_resp_set_status() - for setting the HTTP status string,
- httpd_resp_set_type() - for setting the Content Type,
- httpd_resp_set_hdr() - for appending any additional field value entries in the response header

**备注:**
- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, the request has been responded to.
- No additional data can then be sent for the request.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.
### esp_err_t httpd_resp_send_chunk (httpd_req_t *r, const char *buf, ssize_t buf_len)

This API sends one HTTP chunk.

This API will send the data as an HTTP response to the request. It will use chunked-encoding and send the response in the form of chunks. If you have the entire response contained in a single buffer, please use `httpd_resp_send()` instead.

If no status code and content-type were set, by default this will send 200 OK status code and content type as text/html. You may call the following functions before this API to configure the response headers:

- `httpd_resp_set_status()` - for setting the HTTP status string,
- `httpd_resp_set_type()` - for setting the Content Type,
- `httpd_resp_set_hdr()` - for appending any additional field-value entries in the response header.

**Note:**
- This API is supposed to be called only from the context of a URI handler where `httpd_req_t` request pointer is valid.
- When you are finished sending all your chunks, you must call this function with `buf_len` as 0.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

### esp_err_t httpd_resp_sendstr (httpd_req_t *r, const char *str)

This API simply calls `httpd_resp_send` with buffer length set to string length assuming the buffer contains a null terminated string.
static inline esp_err_t httpd_resp_sendstr_chunk (httpd_req_t *r, const char *str)

API to send a string as an HTTP response chunk.

This API simply calls http_resp_send_chunk with buffer length set to string length assuming the buffer contains a null terminated string

**参数**
- `r` – [in] The request being responded to
- `str` – [in] String to be sent as response body (NULL to finish response packet)

**返回**
- ESP_OK: On successfully sending the response packet
- ESP_ERR_INVALID_ARG: Null request pointer
- ESP_ERR_HTTPD_RESP_HDR: Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND: Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ: Invalid request

**esp_err_t httpd_resp_set_status (httpd_req_t *r, const char *status)**

API to set the HTTP status code.

This API sets the status of the HTTP response to the value specified. By default, the ‘200 OK’ response is sent as the response.

**备注:**
- This API is supposed to be called only from the context of a URI handler where httpd_req_t request pointer is valid.
- This API only sets the status to this value. The status isn’t sent out until any of the send APIs is executed.
- Make sure that the lifetime of the status string is valid till send function is called.

**参数**
- `r` – [in] The request being responded to
- `status` – [in] The HTTP status code of this response

**返回**
- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Null arguments
- ESP_ERR_HTTPD_INVALID_REQ: Invalid request pointer

**esp_err_t httpd_resp_set_type (httpd_req_t *r, const char *type)**

API to set the HTTP content type.

This API sets the ‘Content Type’ field of the response. The default content type is ‘text/html’.

**备注:**
- This API is supposed to be called only from the context of a URI handler where httpd_req_t request pointer is valid.
- This API only sets the content type to this value. The type isn’t sent out until any of the send APIs is executed.
- Make sure that the lifetime of the type string is valid till send function is called.

**参数**
- `r` – [in] The request being responded to
- `type` – [in] The Content Type of the response

**返回**
- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Null arguments
- ESP_ERR_HTTPD_INVALID_REQ: Invalid request pointer


## esp_err_t httpd_resp_set_hdr (httpd_req_t *r, const char *field, const char *value)

API to append any additional headers.

This API sets any additional header fields that need to be sent in the response.

### 备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- The header isn’t sent out until any of the send APIs is executed.
- The maximum allowed number of additional headers is limited to value of max_resp_headers in config structure.
- Make sure that the lifetime of the field value strings are valid till send function is called.

### 参数

- `r` –[in] The request being responded to
- `field` –[in] The field name of the HTTP header
- `value` –[in] The value of this HTTP header

### 返回

- ESP_OK : On successfully appending new header
- ESP_ERR_HTTPD_RESP_HDR : Total additional headers exceed max allowed
- ESP_OK : On successfully appending new header

## esp_err_t httpd_resp_send_err (httpd_req_t *req, httpd_err_code_t error, const char *msg)

For sending out error code in response to HTTP request.

### 备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.
- If you wish to send additional data in the body of the response, please use the lower-level functions directly.

### 参数

- `req` –[in] Pointer to the HTTP request for which the response needs to be sent
- `error` –[in] Error type to send
- `msg` –[in] Error message string (pass NULL for default message)

### 返回

- ESP_OK : On successfully sending the response packet
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

## static inline esp_err_t httpd_resp_send_404 (httpd_req_t *r)

Helper function for HTTP 404.

Send HTTP 404 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

### 备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
Chapter 2. API 参考

- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

### 参数 r [in] The request being responded to

- ESP_OK: On successfully sending the response packet
- ESP_ERR_INVALID_ARG: Null arguments
- ESP_ERR_HTTPD_RESP_SEND: Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ: Invalid request pointer

static inline esp_err_t httpd_resp_send_408(httpd_req_t *r)

Helper function for HTTP 408.

Send HTTP 408 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

### 备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

### 参数 r [in] The request being responded to

- ESP_OK: On successfully sending the response packet
- ESP_ERR_INVALID_ARG: Null arguments
- ESP_ERR_HTTPD_RESP_SEND: Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ: Invalid request pointer

static inline esp_err_t httpd_resp_send_500(httpd_req_t *r)

Helper function for HTTP 500.

Send HTTP 500 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

### 备注:

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

### 参数 r [in] The request being responded to

- ESP_OK: On successfully sending the response packet
- ESP_ERR_INVALID_ARG: Null arguments
- ESP_ERR_HTTPD_RESP_SEND: Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ: Invalid request pointer

int httpd_send(httpd_req_t *r, const char *buf, size_t buf_len)

Raw HTTP send.

Call this API if you wish to construct your custom response packet. When using this, all essential header, eg. HTTP version, Status Code, Content Type and Length, Encoding, etc. will have to be constructed manually,
and HTTP delimiters (CRLF) will need to be placed correctly for separating sub-sections of the HTTP response packet.

If the send override function is set, this API will end up calling that function eventually to send data out.

备注:
- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Unless the response has the correct HTTP structure (which the user must now ensure) it is not guaranteed that it will be recognized by the client. For most cases, you wouldn’t have to call this API, but you would rather use either of: httpd_resp_send(), httpd_resp_send_chunk()

参数
- r [-in] The request being responded to
- buf [-in] Buffer from where the fully constructed packet is to be read
- buf_len [-in] Length of the buffer

返回
- Bytes : Number of bytes that were sent successfully
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

int httpd_socket_send (http_handle_t hd, int sockfd, const char* buf, size_t buf_len, int flags)
A low level API to send data on a given socket

备注: This API is not recommended to be used in any request handler. Use this only for advanced use cases, wherein some asynchronous data is to be sent over a socket.

参数
- hd [-in] server instance
- sockfd [-in] session socket file descriptor
- buf [-in] buffer with bytes to send
- buf_len [-in] data size
- flags [-in] flags for the send() function

返回
- Bytes : The number of bytes sent successfully
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

int httpd_socket_recv (http_handle_t hd, int sockfd, char* buf, size_t buf_len, int flags)
A low level API to receive data from a given socket

This internally calls the default recv function, or the function registered by httpd_sess_set_recv_override().

备注: This API is not recommended to be used in any request handler. Use this only for advanced use cases, wherein some asynchronous communication is required.
Chapter 2. API

- **hd** - [in] server instance
- **sockfd** - [in] session socket file descriptor
- **buf** - [in] buffer with bytes to send
- **buf_len** - [in] data size
- **flags** - [in] flags for the send() function

返回
- Bytes : The number of bytes received successfully
- 0 : Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket recv()

**esp_err_t httpd_register_err_handler** (*httpd_handle_t* handle, *httpd_err_code_t* error, *httpd_err_handler_func_t* handler_fn)

Function for registering HTTP error handlers.

This function maps a handler function to any supported error code given by *httpd_err_code_t*. See prototype *httpd_err_handler_func_t* above for details.

参数
- **handle** - [in] HTTP server handle
- **error** - [in] Error type
- **handler_fn** - [in] User implemented handler function (Pass NULL to unset any previously set handler)

返回
- **ESP_OK** : handler registered successfully
- **ESP_ERR_INVALID_ARG** : invalid error code or server handle

**esp_err_t httpd_start** (*httpd_handle_t* *handle*, const *httpd_config_t* *config*)

Starts the web server.

Create an instance of HTTP server and allocate memory/resources for it depending upon the specified configuration.

Example usage:

```c
// Function for starting the webserver
httpd_handle_t start_webserver(void)
{
    // Generate default configuration
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    // Empty handle to http_server
    httpd_handle_t server = NULL;

    // Start the httpd server
    if (httpd_start(&server, &config) == ESP_OK) {
        // Register URI handlers
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_post);
    }
    // If server failed to start, handle will be NULL
    return server;
}
```

参数
- **config** - [in] Configuration for new instance of the server
- **handle** - [in] Handle to newly created instance of the server. NULL on error

返回
- **ESP_OK** : Instance created successfully
- **ESP_ERR_INVALID_ARG** : Null argument(s)
**Chapter 2. API 参考**

- **ESP_ERR_HTTPD_ALLOC_MEM**: Failed to allocate memory for instance
- **ESP_ERR_HTTPD_TASK**: Failed to launch server task

### esp_err_t httpd_stop (httpd_handle_t handle)

Stops the web server.

Deallocates memory/resources used by an HTTP server instance and deletes it. Once deleted the handle can no longer be used for accessing the instance.

Example usage:

```c
void stop_webserver(httpd_handle_t server)
{
    // Ensure handle is non NULL
    if (server != NULL) {
        // Stop the httpd server
        httpd_stop(server);
    }
}
```

**参数 handle [in]** Handle to server returned by httpd_start

**返回**

- **ESP_OK**: Server stopped successfully
- **ESP_ERR_INVALID_ARG**: Handle argument is null

### esp_err_t httpd_queue_work (httpd_handle_t handle, httpd_work_fn_t work, void *arg)

Queue execution of a function in HTTPD’s context.

This API queues a work function for asynchronous execution.

**备注**: Some protocols require that the web server generate some asynchronous data and send it to the persistently opened connection. This facility is for use by such protocols.

**参数**

- **handle [in]** Handle to server returned by httpd_start
- **work [in]** Pointer to the function to be executed in the HTTPD’s context
- **arg [in]** Pointer to the arguments that should be passed to this function

**返回**

- **ESP_OK**: On successfully queueing the work
- **ESP_FAIL**: Failure in ctrl socket
- **ESP_ERR_INVALID_ARG**: Null arguments

### void *httpd_sess_get_ctx (httpd_handle_t handle, int sockfd)

Get session context from socket descriptor.

Typically if a session context is created, it is available to URI handlers through the httpd_req_t structure. But, there are cases where the web server’s send/receive functions may require the context (for example, for accessing keying information etc). Since the send/receive function only have the socket descriptor at their disposal, this API provides them with a way to retrieve the session context.

**参数**

- **handle [in]** Handle to server returned by httpd_start
- **sockfd [in]** The socket descriptor for which the context should be extracted.

**返回**

- **void**: Pointer to the context associated with this session
- **NULL**: Empty context / Invalid handle / Invalid socket fd
Chapter 2. API 参考

void httpd_sess_set_ctx (httpd_handle_t handle, int sockfd, void *ctx, httpd_free_ctx_fn_t free_fn)

Set session context by socket descriptor.

参数
• handle [in] Handle to server returned by httpd_start
• sockfd [in] The socket descriptor for which the context should be extracted.
• ctx [in] Context object to assign to the session
• free_fn [in] Function that should be called to free the context

void *httpd_sess_get_transport_ctx (httpd_handle_t handle, int sockfd)

Get session ‘transport’ context by socket descriptor.

This context is used by the send/receive functions, for example to manage SSL context.

参见:
httpd_sess_get_ctx()

参数
• handle [in] Handle to server returned by httpd_start
• sockfd [in] The socket descriptor for which the context should be extracted.

返回
• void*: Pointer to the transport context associated with this session
• NULL: Empty context / Invalid handle / Invalid socket fd

void httpd_sess_set_transport_ctx (httpd_handle_t handle, int sockfd, void *ctx, httpd_free_ctx_fn_t free_fn)

Set session ‘transport’ context by socket descriptor.

参见:
httpd_sess_set_ctx()

参数
• handle [in] Handle to server returned by httpd_start
• sockfd [in] The socket descriptor for which the context should be extracted.
• ctx [in] Transport context object to assign to the session
• free_fn [in] Function that should be called to free the transport context

void *httpd_get_global_user_ctx (httpd_handle_t handle)

Get HTTPD global user context (it was set in the server config struct)

参数 handle [in] Handle to server returned by httpd_start
返回 global user context

void *httpd_get_global_transport_ctx (httpd_handle_t handle)

Get HTTPD global transport context (it was set in the server config struct)

参数 handle [in] Handle to server returned by httpd_start
返回 global transport context

esp_err_t httpd_sess_trigger_close (httpd_handle_t handle, int sockfd)

Trigger an httpd session close externally.

备注: Calling this API is only required in special circumstances wherein some application requires to close an httpd client session asynchronously.

参数
Chapter 2. API

- **handle** - [in] Handle to server returned by `httpd_start`
- **sockfd** - [in] The socket descriptor of the session to be closed

**返回**
- **ESP_OK**: On successfully initiating closure
- **ESP_FAIL**: Failure to queue work
- **ESP_ERR_NOT_FOUND**: Socket fd not found
- **ESP_ERR_INVALID_ARG**: Null arguments

```c
esp_err_t httpd_sess_update_lru_counter(httpd_handle_t handle, int sockfd)
```

Update LRU counter for a given socket.

LRU Counters are internally associated with each session to monitor how recently a session exchanged traffic. When LRU purge is enabled, if a client is requesting for connection but maximum number of sockets/sessions is reached, then the session having the earliest LRU counter is closed automatically.

Updating the LRU counter manually prevents the socket from being purged due to the Least Recently Used (LRU) logic, even though it might not have received traffic for some time. This is useful when all open sockets/session are frequently exchanging traffic but the user specifically wants one of the sessions to be kept open, irrespective of when it last exchanged a packet.

**备注**: Calling this API is only necessary if the LRU Purge Enable option is enabled.

---

```c
esp_err_t httpd_get_client_list(httpd_handle_t handle, size_t *fds, int *client_fds)
```

Returns list of current socket descriptors of active sessions.

**备注**: Size of provided array has to be equal or greater then maximum number of opened sockets, configured upon initialization with max_open_sockets field in `httpd_config_t` structure.

---

**参数**
- **handle** - [in] Handle to server returned by `httpd_start`
- **sockfd** - [in] The socket descriptor of the session for which LRU counter is to be updated

**返回**
- **ESP_OK**: Socket found and LRU counter updated
- **ESP_ERR_NOT_FOUND**: Socket not found
- **ESP_ERR_INVALID_ARG**: Null arguments

```c
httpd_sess_update_lru_counter(handle, sockfd)
```

Returns list of current socket descriptors of active sessions.

---

**备注**: Size of provided array has to be equal or greater then maximum number of opened sockets, configured upon initialization with max_open_sockets field in `httpd_config_t` structure.

---

**参数**
- **handle** - [in] Handle to server returned by `httpd_start`
- **fds** - [inout] In: Size of provided client_fds array Out: Number of valid client fds returned in client_fds,
- **client_fds** - [out] Array of client fds

**返回**
- **ESP_OK**: Successfully retrieved session list
- **ESP_ERR_INVALID_ARG**: Wrong arguments or list is longer than provided array

**Structures**

```c
struct httpd_config
```

HTTP Server Configuration Structure.

**备注**: Use `HTTPD_DEFAULT_CONFIG()` to initialize the configuration to a default value and then modify only those fields that are specifically determined by the use case.
Public Members

unsigned **task_priority**
Priority of FreeRTOS task which runs the server

size_t **stack_size**
The maximum stack size allowed for the server task

BaseType_t **core_id**
The core the HTTP server task will run on

uint16_t **server_port**
TCP Port number for receiving and transmitting HTTP traffic

uint16_t **ctrl_port**
UDP Port number for asynchronously exchanging control signals between various components of the server

uint16_t **max_opensockets**
Max number of sockets/clients connected at any time (3 sockets are reserved for internal working of the HTTP server)

uint16_t **max_uri_handlers**
Maximum allowed uri handlers

uint16_t **max_resp_headers**
Maximum allowed additional headers in HTTP response

uint16_t **backlog_conn**
Number of backlog connections

bool **lru_purge_enable**
Purge “Least Recently Used” connection

uint16_t **recv_wait_timeout**
Timeout for recv function (in seconds)

uint16_t **send_wait_timeout**
Timeout for send function (in seconds)

void **global_user_ctx**
Global user context.
This field can be used to store arbitrary user data within the server context. The value can be retrieved using the server handle, available e.g. in the httpd_req_t struct.
When shutting down, the server frees up the user context by calling free() on the global_user_ctx field.
If you wish to use a custom function for freeing the global user context, please specify that here.

```
httpd_free_ctx_fn_t **global_user_ctx_free_fn**
Free function for global user context
```
void *global_transport_ctx

Global transport context.

Similar to global_user_ctx, but used for session encoding or encryption (e.g. to hold the SSL context). It will be freed using free(), unless global_transport_ctx_free_fn is specified.

httpd_free_ctx_fn_t global_transport_ctx_free_fn

Free function for global transport context

bool enable_so_linger

bool to enable/disable linger

int linger_timeout

linger timeout (in seconds)

bool keep_alive_enable

Enable keep-alive timeout

int keep_alive_idle

Keep-alive idle time. Default is 5 (second)

int keep_alive_interval

Keep-alive interval time. Default is 5 (second)

int keep_alive_count

Keep-alive packet retry send count. Default is 3 counts

httpd_open_func_t open_fn

Custom session opening callback.

Called on a new session socket just after accept(), but before reading any data.

This is an opportunity to set up e.g. SSL encryption using global_transport_ctx and the send/recv/pending session overrides.

If a context needs to be maintained between these functions, store it in the session using httpd_sess_set_transport_ctx() and retrieve it later with httpd_sess_get_transport_ctx().

Returning a value other than ESP_OK will immediately close the new socket.

httpd_close_func_t close_fn

Custom session closing callback.

Called when a session is deleted, before freeing user and transport contexts and before closing the socket.

This is a place for custom de-init code common to all sockets.

The server will only close the socket if no custom session closing callback is set. If a custom callback is used, close(sockfd) should be called in here for most cases.

Set the user or transport context to NULL if it was freed here, so the server does not try to free it again.

This function is run for all terminated sessions, including sessions where the socket was closed by the network stack - that is, the file descriptor may not be valid anymore.
httpd_uri_match_func_t uri_match_fn

URI matcher function.

Called when searching for a matching URI: 1) whose request handler is to be executed right after an HTTP request is successfully parsed 2) in order to prevent duplication while registering a new URI handler using httpd_register_uri_handler()

Available options are: 1) NULL : Internally do basic matching using strncmp() 2) httpd_uri_match_wildcard() : URI wildcard matcher

Users can implement their own matching functions (See description of the httpd_uri_match_func_t function prototype)

struct httpd_req

HTTP Request Data Structure.

Public Members

httpd_handle_t handle

Handle to server instance

int method

The type of HTTP request, -1 if unsupported method

const char uri[HTTPD_MAX_URI_LEN + 1]

The URI of this request (1 byte extra for null termination)

size_t content_len

Length of the request body

void *aux

Internally used members

void *user_ctx

User context pointer passed during URI registration.

void *sess_ctx

Session Context Pointer

A session context. Contexts are maintained across ‘sessions’ for a given open TCP connection. One session could have multiple request responses. The web server will ensure that the context persists across all these request and responses.

By default, this is NULL. URI Handlers can set this to any meaningful value.

If the underlying socket gets closed, and this pointer is non-NULL, the web server will free up the context by calling free(), unless free_ctx function is set.

httpd_free_ctx_fn_t free_ctx

Pointer to free context hook

Function to free session context

If the web server’s socket closes, it frees up the session context by calling free() on the sess_ctx member.

If you wish to use a custom function for freeing the session context, please specify that here.
bool ignore_sess_ctx_changes
Flag indicating if Session Context changes should be ignored

By default, if you change the sess_ctx in some URI handler, the http server will internally free the earlier context (if non NULL), after the URI handler returns. If you want to manage the allocation/reallocation/freeing of sess_ctx yourself, set this flag to true, so that the server will not perform any checks on it. The context will be cleared by the server (by calling free_ctx or free()) only if the socket gets closed.

struct httpd_uri
Structure for URI handler.

Public Members

const char *uri
The URI to handle

httpd_method_t method
Method supported by the URI

esp_err_t (*handler)(httpd_req_t *r)
Handler to call for supported request method. This must return ESP_OK, or else the underlying socket will be closed.

void *user_ctx
Pointer to user context data which will be available to handler

Macros

HTTPD_MAX_REQ_HDR_LEN

HTTPD_MAX_URI_LEN

HTTPD_SOCK_ERR_FAIL

HTTPD_SOCK_ERR_INVALID

HTTPD_SOCK_ERR_TIMEOUT

HTTPD_200
HTTP Response 200

HTTPD_204
HTTP Response 204

HTTPD_207
HTTP Response 207
HTTPD_400
HTTP Response 400

HTTPD_404
HTTP Response 404

HTTPD_408
HTTP Response 408

HTTPD_500
HTTP Response 500

HTTPD_TYPE_JSON
HTTP Content type JSON

HTTPD_TYPE_TEXT
HTTP Content type text/HTML

HTTPD_TYPE_OCTET
HTTP Content type octet-stream

ESP_HTTPD_DEF_CTRL_PORT
HTTP Server control socket port

HTTPD_DEFAULT_CONFIG()

ESP_ERR_HTTPD_BASE
Starting number of HTTPD error codes

ESP_ERR_HTTPD_HANDLERS_FULL
All slots for registering URI handlers have been consumed

ESP_ERR_HTTPD_HANDLER_EXISTS
URI handler with same method and target URI already registered

ESP_ERR_HTTPD_INVALID_REQ
Invalid request pointer

ESP_ERR_HTTPD_RESULT_TRUNC
Result string truncated

ESP_ERR_HTTPD_RESP_HDR
Response header field larger than supported

ESP_ERR_HTTPD_RESP_SEND
Error occurred while sending response packet

ESP_ERR_HTTPD_ALLOC_MEM
Failed to dynamically allocate memory for resource
Chapter 2. API

**ESP_ERR_HTTPD_TASK**

Failed to launch server task/thread

**HTTPD_RESP_USE_STRLEN**

**Type Definitions**

typedef struct httpd_req httpd_req_t

HTTP Request Data Structure.

typedef struct httpd_uri httpd_uri_t

Structure for URI handler.

typedef int (*httpd_send_func_t)(httpd_handle_t hd, int sockfd, const char *buf, size_t buf_len, int flags)

Prototype for HTTPDs low-level send function.

备注：User specified send function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will eventually be conveyed as return value of httpd_send() function

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hd</td>
<td>[in] server instance</td>
</tr>
<tr>
<td>sockfd</td>
<td>[in] session socket file descriptor</td>
</tr>
<tr>
<td>buf</td>
<td>[in] buffer with bytes to send</td>
</tr>
<tr>
<td>buf_len</td>
<td>[in] data size</td>
</tr>
<tr>
<td>flags</td>
<td>[in] flags for the send() function</td>
</tr>
</tbody>
</table>

Return

- Bytes : The number of bytes sent successfully
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

typedef int (*httpd_recv_func_t)(httpd_handle_t hd, int sockfd, char *buf, size_t buf_len, int flags)

Prototype for HTTPDs low-level recv function.

备注：User specified recv function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will eventually be conveyed as return value of httpd_req_recv() function

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hd</td>
<td>[in] server instance</td>
</tr>
<tr>
<td>sockfd</td>
<td>[in] session socket file descriptor</td>
</tr>
<tr>
<td>buf</td>
<td>[in] buffer with bytes to send</td>
</tr>
<tr>
<td>buf_len</td>
<td>[in] data size</td>
</tr>
<tr>
<td>flags</td>
<td>[in] flags for the send() function</td>
</tr>
</tbody>
</table>

Return

- Bytes : The number of bytes received successfully
- 0 : Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket recv()
typedef int (*httpd_pending_func_t)(httpd_handle_t hd, int sockfd)
Prototype for HTTPD's low-level “get pending bytes” function.

备注: User specified pending function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will be handled accordingly in the server task.

Param hd [in] server instance
Param sockfd [in] session socket file descriptor
Return
- Bytes : The number of bytes waiting to be received
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket pending()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket pending()

typedef esp_err_t (*httpd_err_handler_func_t)(httpd_req_t *req, httpd_err_code_t error)
Function prototype for HTTP error handling.
This function is executed upon HTTP errors generated during internal processing of an HTTP request. This is used to override the default behavior on error, which is to send HTTP error response and close the underlying socket.

备注:
- If implemented, the server will not automatically send out HTTP error response codes, therefore, httpd_resp_send_err() must be invoked inside this function if user wishes to generate HTTP error responses.
- When invoked, the validity of uri, method, content_len and user_ctx fields of the httpd_req_t parameter is not guaranteed as the HTTP request may be partially received/parsed.
- The function must return ESP_OK if underlying socket needs to be kept open. Any other value will ensure that the socket is closed. The return value is ignored when error is of type HTTPD_500_INTERNAL_SERVER_ERROR and the socket closed anyway.

Param req [in] HTTP request for which the error needs to be handled
Param error [in] Error type
Return
- ESP_OK : error handled successful
- ESP_FAIL : failure indicates that the underlying socket needs to be closed

typedef void *httpd_handle_t
HTTP Server Instance Handle.
Every instance of the server will have a unique handle.

typedef enum http_method httpd_method_t
HTTP Method Type wrapper over “enum http_method” available in “http_parser” library.

typedef void (*httpd_free_ctx_fn_t)(void *ctx)
Prototype for freeing context data (if any)

Param ctx [in] object to free
typedef esp_err_t (*httpd_open_func_t)(httpd_handle_t hd, int sockfd)

Function prototype for opening a session.

Called immediately after the socket was opened to set up the send/recv functions and other parameters of the socket.

**Param** hd **[in]** server instance

**Param** sockfd **[in]** session socket file descriptor

**Return**
- ESP_OK: On success
- Any value other than ESP_OK will signal the server to close the socket immediately

typedef void (*httpd_close_func_t)(httpd_handle_t hd, int sockfd)

Function prototype for closing a session.

备注: It’s possible that the socket descriptor is invalid at this point, the function is called for all terminated sessions. Ensure proper handling of return codes.

**Param** hd **[in]** server instance

**Param** sockfd **[in]** session socket file descriptor

typedef bool (*httpd_uri_match_func_t)(const char *reference_uri, const char *uri_to_match, size_t match_upto)

Function prototype for URI matching.

**Param** reference_uri **[in]** URI/template with respect to which the other URI is matched

**Param** uri_to_match **[in]** URI/template being matched to the reference URI/template

**Param** match_upto **[in]** For specifying the actual length of uri_to_match up to which the matching algorithm is to be applied (The maximum value is strlen(uri_to_match), independent of the length of reference_uri)

**Return** true on match

typedef struct httpd_config httpd_config_t

HTTP Server Configuration Structure.

备注: Use HTTPD_DEFAULT_CONFIG() to initialize the configuration to a default value and then modify only those fields that are specifically determined by the use case.

typedef void (*httpd_work_fn_t)(void *arg)

Prototype of the HTTPD work function Please refer to httpd_queue_work() for more details.

**Param** arg **[in]** The arguments for this work function

Enumerations

enum httpd_err_code_t

Error codes sent as HTTP response in case of errors encountered during processing of an HTTP request.

**Values:**

enumerator HTTPD_500_INTERNAL_SERVER_ERROR
2.2.10 HTTPS 服务器

概述

HTTPS 服务器组件建立在 HTTP 服务器组件的基础上。该服务器借助常规 HTTP 服务器中的钩子注册函数，注册 SSL 会话回调处理函数。

HTTP 服务器 组件的所有文档同样适用于用户按照本文档搭建的服务器。

API 说明

下列 HTTP 服务器 的 API 已不适用于 HTTPS 服务器。这些 API 仅限内部使用，用于处理安全会话和维护内部状态。

- “send”, “receive” 和 “pending” 回调注册函数——处理安全套接字
  - httpd_sess_set_send_override()
  - httpd_sess_set_recv_override()
  - httpd_sess_set_pending_override()

- “transport context”——传输层上下文
  - httpd_sess_get_transport_ctx()：返回会话使用的 SSL
  - httpd_sess_set_transport_ctx()
  - httpd_get_global_transport_ctx()：返回共享的 SSL 上下文
  - httpd_config::global_transport_ctx
  - httpd_config::global_transport_ctx_free_fn
Chapter 2. API 参考

- `httpd_config::open_fn`：用于设置安全套接字

其他 API 均可使用，没有其他限制。

如何使用

请参考示例 `protocols/https_server` 来学习如何搭建安全的服务器。

总体而言，您只需要生成证书，将其嵌入到固件中，并且在初始化结构体中配置好正确的证书地址和长度后，将其传入服务器启动函数。

通过改变初始化配置结构体中的标志 `httpd_ssl_config::transport_mode`，可以选择是否需要 SSL 连接来启动服务器。在测试时或在速度比安全性更重要的可信环境中，您可以使用此功能。

性能

建立起始会议大约需要两秒。在时钟速度较慢或日志记录冗余信息较多的情况下，可能需要花费更多时间。后续通过已打开的安全套接字建立请求的速度会更快，最快只需不到 100 ms。

API 参考

Header File

- components/esp_https_server/include/esp_https_server.h

Functions

```c
esp_err_t httpd_ssl_start (httpd_handle_t *handle, httpd_ssl_config_t *config)
```

Create a SSL capable HTTP server (secure mode may be disabled in config)

参数

- `config` [inout] - server config, must not be const. Does not have to stay valid after calling this function.
- `handle` [out] - storage for the server handle, must be a valid pointer

返回 success

```c
esp_err_t httpd_ssl_stop (httpd_handle_t handle)
```

Stop the server. Blocks until the server is shut down.

参数 `handle` [in]

返回

- ESP_OK: Server stopped successfully
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_FAIL: Failure to shut down server

Structures

```c
struct esp_https_server_user_cb_arg
```

Callback data struct, contains the ESP-TLS connection handle and the connection state at which the callback is executed.

Public Members

```c
httpd_ssl_user_cb_state_t user_cb_state
```

State of user callback
Esp-TLS connection handle

struct \textit{httpd_ssl_config}
\begin{verbatim}
HTTPS server config struct
Please use HTTPD_SSL_CONFIG_DEFAULT() to initialize it.
\end{verbatim}

Public Members

\begin{verbatim}
httpd_config_t \textit{httpd}
\begin{verbatim}
Underlying HTTPD server config
Parameters like task stack size and priority can be adjusted here.
\end{verbatim}
\end{verbatim}

const uint8_t* \textit{servercert}
\begin{verbatim}
Server certificate
\end{verbatim}

size_t \textit{servercert_len}
\begin{verbatim}
Server certificate byte length
\end{verbatim}

const uint8_t* \textit{cacert_pem}
\begin{verbatim}
CA certificate ((CA used to sign clients, or client cert itself)
\end{verbatim}

size_t \textit{cacert_len}
\begin{verbatim}
CA certificate byte length
\end{verbatim}

const uint8_t* \textit{prvtkey_pem}
\begin{verbatim}
Private key
\end{verbatim}

size_t \textit{prvtkey_len}
\begin{verbatim}
Private key byte length
\end{verbatim}

\begin{verbatim}
httpd_ssl_transport_mode_t \textit{transport_mode}
\begin{verbatim}
Transport Mode (default secure)
\end{verbatim}
\end{verbatim}

uint16_t \textit{port_secure}
\begin{verbatim}
Port used when transport mode is secure (default 443)
\end{verbatim}

uint16_t \textit{port_insecure}
\begin{verbatim}
Port used when transport mode is insecure (default 80)
\end{verbatim}

bool \textit{session_tickets}
\begin{verbatim}
Enable tls session tickets
\end{verbatim}

bool \textit{use_secure_element}
\begin{verbatim}
Enable secure element for server session
\end{verbatim}

\begin{verbatim}
esp_https_server_user_cb \textit{*user_cb}
\begin{verbatim}
User callback for esp_https_server
\end{verbatim}
\end{verbatim}
void *ssl_userdata
user data to add to the ssl context

esp_tls_handshake_callback cert_select_cb
Certificate selection callback to use

const char **alpn_protos
Application protocols the server supports in order of preference. Used for negotiating during the TLS handshake, first one the client supports is selected. The data structure must live as long as the https server itself!

Macros
HTTPD_SSL_CONFIG_DEFAULT()
Default config struct init
(http_server default config had to be copied for customization)

Notes:
• port is set when starting the server, according to `transport_mode`
• one socket uses ~ 40k BRAM with SSL, we reduce the default socket count to 4
• SSL sockets are usually long-lived, closing LRU prevents pool exhaustion DOS
• Stack size may need adjustments depending on the user application

Type Definitions
typedef struct esp_https_server_user_cb_arg esp_https_server_user_cb_arg_t
Callback data struct, contains the ESP-TLS connection handle and the connection state at which the callback is executed.
typedef void esp_https_server_user_cb (esp_https_server_user_cb_arg_t *user_cb)
Callback function prototype Can be used to get connection or client information (SSL context) E.g. Client certificate, Socket FD, Connection state, etc.

    Param user_cb Callback data struct
typedef struct httpd_ssl_config httpd_ssl_config_t

Enumerations
eenum httpd_ssl_transport_mode_t
Values:

    enumerator HTTPD_SSL_TRANSPORT_SECURE

    enumerator HTTPD_SSL_TRANSPORT_INSECURE
eenum httpd_ssl_user_cb_state_t
Indicates the state at which the user callback is executed, i.e at session creation or session close.
Values:

    enumerator HTTPD_SSL_USER_CB_SESS_CREATE

    enumerator HTTPD_SSL_USER_CB_SESS_CLOSE
2.2.11 ICMP Echo

Overview

ICMP (Internet Control Message Protocol) is used for diagnostic or control purposes or generated in response to errors in IP operations. The common network utility ping is implemented based on the ICMP packets with the type field value of 0, also called Echo Reply.

During a ping session, the source host firstly sends out an ICMP echo request packet and wait for an ICMP echo reply with specific times. In this way, it also measures the round-trip time for the messages. After receiving a valid ICMP echo reply, the source host will generate statistics about the IP link layer (e.g. packet loss, elapsed time, etc).

It is common that IoT device needs to check whether a remote server is alive or not. The device should show the warnings to users when it got offline. It can be achieved by creating a ping session and sending/parsing ICMP echo packets periodically.

To make this internal procedure much easier for users, ESP-IDF provides some out-of-box APIs.

Create a new ping session  To create a ping session, you need to fill in the esp_ping_config_t configuration structure firstly, specifying target IP address, interval times, and etc. Optionally, you can also register some callback functions with the esp_ping_callbacks_t structure.

Example method to create a new ping session and register callbacks:

```c
static void test_on_ping_success(esp_ping_handle_t hdl, void *args)
{
    // optionally, get callback arguments
    // const char* str = (const char*) args;
    // printf("%s\r\n", str); // "foo"
    uint8_t ttl;
    uint16_t seqno;
    uint32_t elapsed_time, recv_len;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TTL, &ttl, sizeof(ttl));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr, sizeof(target_addr));
    esp_ping_get_profile(hdl, ESP_PING_PROF_SIZE, &recv_len, sizeof(recv_len));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TIMEGAP, &elapsed_time, sizeof(elapsed_time));
    printf("%d bytes from %s icmp_seq=%d ttl=%d time=%d ms\n", recv_len, inet_ntoa(target_addr.u_addr.ip4), seqno, ttl, elapsed_time);
}

static void test_on_ping_timeout(esp_ping_handle_t hdl, void *args)
{
    uint16_t seqno;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr, sizeof(target_addr));
    printf("From %s icmp_seq=%d timeout\n", inet_ntoa(target_addr.u_addr.ip4), seqno);
}

static void test_on_ping_end(esp_ping_handle_t hdl, void *args)
{
    uint32_t transmitted;
    uint32_t received;
    uint32_t total_time_ms;
    esp_ping_get_profile(hdl, ESP_PING_PROF_REQUEST, &transmitted,
    esp_ping_get_profile(hdl, ESP_PING_PROF_TIMEGAP, &total_time_ms, sizeof(total_time_ms));
    printf("Total transmitted = %d packets in %d ms\n", transmitted, total_time_ms);
}
```
void initialize_ping()
{
    /* convert URL to IP address */
    ip_addr_t target_addr;
    struct addrinfo hint;
    struct addrinfo *res = NULL;
    memset(&hint, 0, sizeof(hint));
    memset(&target_addr, 0, sizeof(target_addr));
    getaddrinfo("www.espressif.com", NULL, &hint, &res);
    struct in_addr addr4 = ((struct sockaddr_in *) (res->ai_addr))->sin_addr;
    inet_addr_to_ip4addr(ip_2_ip4(&target_addr), &addr4);
    freeaddrinfo(res);

    esp_ping_config_t ping_config = ESP_PING_DEFAULT_CONFIG();
    ping_config.target_addr = target_addr; // target IP address
    ping_config.count = ESP_PING_COUNT_INFINITE; // ping in infinite mode, esp_ping_stop can stop it

    /* set callback functions */
    esp_ping_callbacks_t cbs;
    cbs.on_ping_success = test_on_ping_success;
    cbs.on_ping_timeout = test_on_ping_timeout;
    cbs.on_ping_end = test_on_ping_end;
    cbs.cb_args = "foo"; // arguments that will feed to all callback functions, can be NULL
    cbs.cb_args = eth_event_group;

    esp_ping_handle_t ping;
    esp_ping_new_session(&ping_config, &cbs, &ping);
}

Start and Stop ping session You can start and stop ping session with the handle returned by esp_ping_new_session. Note that, the ping session won’t start automatically after creation. If the ping session is stopped, and restart again, the sequence number in ICMP packets will recount from zero again.

Delete a ping session If a ping session won’t be used any more, you can delete it with esp_ping_delete_session. Please make sure the ping session is in stop state (i.e. you have called esp_ping_stop before or the ping session has finished all the procedures) when you call this function.

Get runtime statistics As the example code above, you can call esp_ping_get_profile to get different runtime statistics of ping session in the callback function.

Application Example

ICMP echo example: protocols/icmp_echo

API Reference

Header File
Chapter 2. API

- components/lwip/include/apps/ping/ping_sock.h

**Functions**

*esp_err_t esp_ping_new_session*(const *esp_ping_config_t* config, const *esp_ping_callbacks_t* cbs, *esp_ping_handle_t* hdl_out)

Create a ping session.

参数
- **config** – ping configuration
- **cbs** – a bunch of callback functions invoked by internal ping task
- **hdl_out** – handle of ping session

返回
- ESP_ERR_INVALID_ARG: invalid parameters (e.g. configuration is null, etc)
- ESP_ERR_NO_MEM: out of memory
- ESP_FAIL: other internal error (e.g. socket error)
- ESP_OK: create ping session successfully, user can take the ping handle to do follow-on jobs

*esp_err_t esp_ping_delete_session*(esp_ping_handle_t hdl)

Delete a ping session.

参数 **hdl** – handle of ping session

返回
- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: delete ping session successfully

*esp_err_t esp_ping_start*(esp_ping_handle_t hdl)

Start the ping session.

参数 **hdl** – handle of ping session

返回
- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: start ping session successfully

*esp_err_t esp_ping_stop*(esp_ping_handle_t hdl)

Stop the ping session.

参数 **hdl** – handle of ping session

返回
- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: stop ping session successfully

*esp_err_t esp_ping_get_profile*(esp_ping_handle_t hdl, esp_ping_profile_t profile, void* data, uint32_t size)

Get runtime profile of ping session.

参数
- **hdl** – handle of ping session
- **profile** – type of profile
- **data** – profile data
- **size** – profile data size

返回
- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_ERR_INVALID_SIZE: the actual profile data size doesn’t match the “size” parameter
- ESP_OK: get profile successfully

**Structures**

struct *esp_ping_callbacks_t*

Type of “ping” callback functions.
Public Members

void *cb_args
arguments for callback functions

void (*on_ping_success)(esp_ping_handle_t hdl, void *args)
Invoked by internal ping thread when received ICMP echo reply packet.

void (*on_ping_timeout)(esp_ping_handle_t hdl, void *args)
Invoked by internal ping thread when receive ICMP echo reply packet timeout.

void (*on_ping_end)(esp_ping_handle_t hdl, void *args)
Invoked by internal ping thread when a ping session is finished.

struct esp_ping_config_t
Type of "ping" configuration.

Public Members

uint32_t count
A "ping" session contains count procedures

uint32_t interval_ms
Milliseconds between each ping procedure

uint32_t timeout_ms
Timeout value (in milliseconds) of each ping procedure

uint32_t data_size
Size of the data next to ICMP packet header

int tos
Type of Service, a field specified in the IP header

int ttl
Time to Live, a field specified in the IP header

ip_addr_t target_addr
Target IP address, either IPv4 or IPv6

uint32_t task_stack_size
Stack size of internal ping task

uint32_t task_prio
Priority of internal ping task

uint32_t interface
Netif index, interface=0 means NETIF_NO_INDEX
Chapter 2. API

**Macros**

`ESP_PING_DEFAULT_CONFIG`()  
Default ping configuration.

`ESP_PING_COUNT_INFINITE`  
Set ping count to zero will ping target infinitely

**Type Definitions**

typedef void *`esp_ping_handle_t`  
Type of “ping” session handle.

**Enumerations**

enum `esp_ping_profile_t`  
Profile of ping session.

  * `ESP_PING_PROF_SEQNO`  
    Sequence number of a ping procedure
  * `ESP_PING_PROF_TOS`  
    Type of service of a ping procedure
  * `ESP_PING_PROF_TTL`  
    Time to live of a ping procedure
  * `ESP_PING_PROF_REQUEST`  
    Number of request packets sent out
  * `ESP_PING_PROF_REPLY`  
    Number of reply packets received
  * `ESP_PING_PROF_IPADDR`  
    IP address of replied target
  * `ESP_PING_PROF_SIZE`  
    Size of received packet
  * `ESP_PING_PROF_TIMEGAP`  
    Elapsed time between request and reply packet
  * `ESP_PING_PROF_DURATION`  
    Elapsed time of the whole ping session

2.2.12 mDNS 服务

mDNS 是一种组播 UDP 服务，用来提供本地网络服务和主机发现。
自 v5.0 版本起，ESP-IDF 组件 mDNS 已从 ESP-IDF 中迁出至独立的仓库：
### 2.2.13 Mbed TLS

**Mbed TLS** is a C library that implements cryptographic primitives, X.509 certificate manipulation and the SSL/TLS and DTLS protocols. Its small code footprint makes it suitable for embedded systems.

Mbed TLS supports SSL 3.0 up to TLS 1.3 and DTLS 1.0 to 1.2 communication by providing the following:

- TCP/IP communication functions: listen, connect, accept, read/write.
- SSL/TLS communication functions: init, handshake, read/write.
- X.509 functions: CRT, CRL and key handling
- Random number generation
- Hashing
- Encryption/decryption

**Note:** ESP-IDF uses a fork of Mbed TLS which includes a few patches (related to hardware routines of certain modules like bignum (MPI) and ECC) over vanilla Mbed TLS.

Mbed TLS is in the process of migrating all the documentation to a single place. In the meantime, users can find the documentation at the [old Mbed TLS site](https://github.com/espressif/esp-mbedTLS).

**Mbed TLS Support in ESP-IDF**

Please find the information about the Mbed TLS versions present in different branches of ESP-IDF here.

**Note:** Please refer the [ESP-IDF Migration Guide](https://github.com/espressif/esp-idf) to migrate from Mbed TLS version 2.x to version 3.0 or greater.

**Application Examples**

Examples in ESP-IDF use **ESP-TLS** which provides a simplified API interface for accessing the commonly used TLS functionality.

Refer to the examples `protocols/https_server/simple` (Simple HTTPS server) and `protocols/https_request` (Make HTTPS requests) for more information.

If the Mbed TLS API is to be used directly, refer to the example `protocols/https_mbedtls`.

**Alternatives**

**ESP-TLS** acts as an abstraction layer over the underlying SSL/TLS library and thus has an option to use Mbed TLS or wolfSSL as the underlying library. By default, only Mbed TLS is available and used in ESP-IDF whereas wolfSSL is available publicly at [https://github.com/espressif/esp-wolfSSL](https://github.com/espressif/esp-wolfSSL) with the upstream submodule pointer.

Please refer to **ESP-TLS: Underlying SSL/TLS Library Options** docs for more information on this and comparison of Mbed TLS and wolfSSL.
Important Config Options

Following is a brief list of important config options accessible at Component Config -> mbedTLS. The full list of config options can be found here.

- `CONFIG_MBEDTLS_SSL_PROTO_TLS1_2`: Support for TLS 1.2
- `CONFIG_MBEDTLS_SSL_PROTO_TLS1_3`: Support for TLS 1.3
- `CONFIG_MBEDTLS_CERTIFICATE_BUNDLE`: Support for trusted root certificate bundle (more about this: [ESP x509 Certificate Bundle](#))
- `CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS`: Support for TLS Session Resumption: Client session tickets
- `CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS`: Support for TLS Session Resumption: Server session tickets
- `CONFIG_MBEDTLS_HARDWARE_SHA`: Support for hardware SHA acceleration
- `CONFIG_MBEDTLS_HARDWARE_AES`: Support for hardware AES acceleration
- `CONFIG_MBEDTLS_HARDWARE_MPI`: Support for hardware MPI (bignum) acceleration

备注: Mbed TLS v3.0.0 and later support only TLS 1.2 and TLS 1.3 (SSL 3.0, TLS 1.0, TLS 1.1 and DTLS 1.0 are not supported). The support for TLS 1.3 is experimental and only supports the client-side. More information about this can be found out here.

Performance and Memory Tweaks

Reducing Heap Usage The following table shows typical memory usage with different configs when the protocols/https_request example (with Server Validation enabled) was run with Mbed TLS as the SSL/TLS library.

<table>
<thead>
<tr>
<th>Mbed Test</th>
<th>TLS Related Configs</th>
<th>Heap Usage (approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>NA</td>
<td>42196 B</td>
</tr>
<tr>
<td>Enable SSL Variable Length</td>
<td><code>CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH</code></td>
<td>42120 B</td>
</tr>
<tr>
<td>Disable Keep Peer Certificate</td>
<td><code>CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE</code></td>
<td>38533 B</td>
</tr>
<tr>
<td>Enable Dynamic TX/RX Buffer</td>
<td><code>CONFIG_MBEDTLS_DYNAMIC_BUFFER</code></td>
<td><code>CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA</code></td>
</tr>
</tbody>
</table>

备注: These values are subject to change with change in configuration options and versions of Mbed TLS.

Reducing Binary Size Under Component Config -> mbedTLS, there are multiple Mbed TLS features which are enabled by default but can be disabled if not needed to save code size. More information can be about this can be found in Minimizing Binary Size docs.

此 API 部分的示例代码存放在 ESP-IDF 示例项目的 protocols 目录下。

2.2.14 IP 网络层协议

IP 网络层协议（应用层协议之下）的文档存放在连网 API 目录下。
2.3 藍牙 API

2.3.1 BT COMMON

BT GENERIC DEFINES

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_bt_defs.h

Structures

struct esp_bt_uuid_t

UUID type.

Public Members

uint16_t len

UUID length, 16bit, 32bit or 128bit

uint16_t uuid16

16bit UUID

uint32_t uuid32

32bit UUID

uint8_t uuid128[ESP_UUID_LEN_128]

128bit UUID

union esp_bt_uuid_t::[anonymous] uuid

UUID

Macros

ESP_BLUEDROID_STATUS_CHECK (status)

ESP_BT_OCTET16_LEN

ESP_BT_OCTET8_LEN

ESP_DEFAULT_GATT_IF

Default GATT interface id.

ESP_BLE_PRIM_ADV_INT_MIN

Minimum advertising interval for undirected and low duty cycle directed advertising
ESP_BLE_PRIM_ADV_INT_MAX
Maximum advertising interval for undirected and low duty cycle directed advertising

ESP_BLE_CONN_INT_MIN
relate to BTM_BLE_CONN_INT_MIN in stack/btm_ble_api.h

ESP_BLE_CONN_INT_MAX
relate to BTM_BLE_CONN_INT_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_LATENCY_MAX
relate to ESP_BLE_CONN_LATENCY_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_SUP_TOUT_MIN
relate to BTM_BLE_CONN_SUP_TOUT_MIN in stack/btm_ble_api.h

ESP_BLE_CONN_SUP_TOUT_MAX
relate to ESP_BLE_CONN_SUP_TOUT_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_PARAM_UNDEF

ESP_BLE_SCAN_PARAM_UNDEF

ESP_BLE_IS_VALID_PARAM (x, min, max)
Check the param is valid or not.

ESP_UUID_LEN_16

ESP_UUID_LEN_32

ESP_UUID_LEN_128

ESP_BD_ADDR_LEN
Bluetooth address length.

ESP_BLE_ENC_KEY_MASK
Used to exchange the encryption key in the init key & response key.

ESP_BLE_ID_KEY_MASK
Used to exchange the IRK key in the init key & response key.

ESP_BLE_CSR_KEY_MASK
Used to exchange the CSRK key in the init key & response key.

ESP_BLE_LINK_KEY_MASK
Used to exchange the link key(this key just used in the BLE & BR/EDR coexist mode) in the init key & response key.

ESP_APP_ID_MIN
Minimum of the application id.
Chapter 2. API

**ESP_APP_ID_MAX**

Maximum of the application id.

**ESP_BD_ADDR_STR**

**ESP_BD_ADDR_HEX** (addr)

### Type Definitions

typedef uint8_t esp_octet16_t[ESP_BT_OCTET16_LEN]

typedef uint8_t esp_octet8_t[ESP_BT_OCTET8_LEN]

typedef uint8_t esp_link_key[ESP_BT_OCTET16_LEN]

typedef uint8_t esp_bd_addr_t[ESP_bd_ADDR_LEN]

Bluetooth device address.

typedef uint8_t esp_ble_key_mask_t

### Enumerations

enum esp_bt_status_t

Status Return Value.

*Values:*

enumerator ESP_BT_STATUS_SUCCESS

enumerator ESP_BT_STATUS_FAIL

enumerator ESP_BT_STATUS_NOT_READY

enumerator ESP_BT_STATUS_NOMEM

enumerator ESP_BT_STATUS_BUSY

enumerator ESP_BT_STATUS_DONE

enumerator ESP_BT_STATUS_UNSUPPORTED

enumerator ESP_BT_STATUS_PARM_INVALID

enumerator ESP_BT_STATUS_UNHANDLED

enumerator ESP_BT_STATUS_AUTH_FAILURE

enumerator ESP_BT_STATUS_RMT_DEV_DOWN
enumerator ESP_BT_STATUS_AUTH_REJECTED
enumerator ESP_BT_STATUS_INVALID_STATIC_RAND_ADDR
enumerator ESP_BT_STATUS_PENDING
enumerator ESP_BT_STATUS_UNACCEPT_CONN_INTERVAL
enumerator ESP_BT_STATUS_PARAM_OUT_OF_RANGE
enumerator ESP_BT_STATUS_TIMEOUT
enumerator ESP_BT_STATUS_PEER_LE_DATA_LEN_UNSUPPORTED
enumerator ESP_BT_STATUS_CONTROL_LE_DATA_LEN_UNSUPPORTED
enumerator ESP_BT_STATUS_ERR_ILLEGAL_PARAMETER_FMT
enumerator ESP_BT_STATUS_MEMORY_FULL
enumerator ESP_BT_STATUS_EIR_TOO_LARGE

enum esp_bt_dev_type_t
  Bluetooth device type.
  
  Values:
  
  enumerator ESP_BT_DEVICE_TYPE_BREDR
  enumerator ESP_BT_DEVICE_TYPE_BLE
  enumerator ESP_BT_DEVICE_TYPE_DUMO

enum esp_ble_addr_type_t
  BLE device address type.
  
  Values:
  
  enumerator BLE_ADDR_TYPE_PUBLIC
  enumerator BLE_ADDR_TYPE_RANDOM
  enumerator BLE_ADDR_TYPE_RPA_PUBLIC
  enumerator BLE_ADDR_TYPE_RPA_RANDOM
enum esp_ble_wl_addr_type_t
   white list address type

   Values:

   enumerator BLE_WL_ADDR_TYPE_PUBLIC

   enumerator BLE_WL_ADDR_TYPE_RANDOM

### BT MAIN API

### API Reference

### Header File

- components/bt/host/bluedroid/api/include/api/esp_bt_main.h

### Functions

**esp_bluedroid_status_t esp_bluedroid_get_status**(void)
   Get bluetooth stack status.
   返回 Bluetooth stack status

**esp_err_t esp_bluedroid_enable**(void)
   Enable bluetooth, must after esp_bluedroid_init().

   返回
   • ESP_OK : Succeed
   • Other : Failed

**esp_err_t esp_bluedroid_disable**(void)
   Disable bluetooth, must prior to esp_bluedroid_deinit().

   返回
   • ESP_OK : Succeed
   • Other : Failed

**esp_err_t esp_bluedroid_init**(void)
   Init and alloc the resource for bluetooth, must be prior to every bluetooth stuff.

   返回
   • ESP_OK : Succeed
   • Other : Failed

**esp_err_t esp_bluedroid_deinit**(void)
   Deinit and free the resource for bluetooth, must be after every bluetooth stuff.

   返回
   • ESP_OK : Succeed
   • Other : Failed

### Enumerations

enum esp_bluedroid_status_t
   Bluetooth stack status type, to indicate whether the bluetooth stack is ready.

   Values:
Chapter 2. API

enumerator **ESP_BLUEDROID_STATUS_UNINITIALIZED**
Bluetooth not initialized

counter **ESP_BLUEDROID_STATUS_INITIALIZED**
Bluetooth initialized but not enabled

counter **ESP_BLUEDROID_STATUS_ENABLED**
Bluetooth initialized and enabled

**BT DEVICE APIs**

**Overview** Bluetooth device reference APIs.

**API Reference**

**Header File**
- components/bt/host/bluedroid/api/include/api/esp_bt_device.h

**Functions**

*const uint8_t* esp_bt_dev_get_address (void)
Get bluetooth device address. Must use after “esp_bluedroid_enable”.
Return bluetooth device address (six bytes), or NULL if bluetooth stack is not enabled

*esp_err_t* esp_bt_dev_set_device_name (const char *name)
Set bluetooth device name. This function should be called after esp_bluedroid_enable() completes successfully.
A BR/EDR/LE device type shall have a single Bluetooth device name which shall be identical irrespective of the physical channel used to perform the name discovery procedure.

参数 name - [in]: devicename to be set
返回
- ESP_OK: Succeed
- ESP_ERR_INVALID_ARG: if nameis NULL pointer or empty, or string length out of limit
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**2.3.2 BT LE**

**GAP API**

**Application Example** Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a SMP security client demo and its tutorial. This demo initiates its security parameters and acts as a GATT client, which can send a security request to the peer device and then complete the encryption procedure.
  - bluetooth/bluedroid/ble/gatt_security_client
  - GATT Security Client Example Walkthrough
- This is a SMP security server demo and its tutorial. This demo initiates its security parameters and acts as a GATT server, which can send a pair request to the peer device and then complete the encryption procedure.
  - bluetooth/bluedroid/ble/gatt_security_server
  - GATT Security Server Example Walkthrough
Chapter 2. API

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_gap_ble_api.h

Functions

`esp_err_t esp_ble_gap_register_callback (esp_gap_ble_cb_t callback)`

This function is called to occur gap event, such as scan result.

**参数**
- `callback` - [in] callback function

**返回**
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_config_adv_data (esp_ble_adv_data_t *adv_data)`

This function is called to override the BTA default ADV parameters.

**参数**
- `adv_data` - [in] Pointer to User defined ADV data structure. This memory space can not be freed until callback of config_adv_data is received.

**返回**
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_set_scan_params (esp_ble_scan_params_t *scan_params)`

This function is called to set scan parameters.

**参数**
- `scan_params` - [in] Pointer to User defined scan_params data structure. This memory space can not be freed until callback of set_scan_params

**返回**
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_start_scanning (uint32_t duration)`

This procedure keep the device scanning the peer device which advertising on the air.

**参数**
- `duration` - [in] Keeping the scanning time, the unit is second.

**返回**
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_stop_scanning (void)`

This function call to stop the device scanning the peer device which advertising on the air.

**返回**
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_start_advertising (esp_ble_adv_params_t *adv_params)`

This function is called to start advertising.

**参数**
- `adv_params` - [in] pointer to User defined adv_params data structure.

**返回**
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_stop_advertising (void)`

This function is called to stop advertising.

**返回**
- ESP_OK: success
- other: failed
**esp_err_t esp_ble_gap_update_conn_params (esp_ble_conn_update_params_t *params)**

Update connection parameters, can only be used when connection is up.

- **params [in]** - connection update parameters
- **返回**
  - ESP_OK: success
  - other: failed

**esp_err_t esp_ble_gap_set_pkt_data_len (esp_bd_addr_t remote_device, uint16_t tx_data_length)**

This function is to set maximum LE data packet size.

- **返回**
  - ESP_OK: success
  - other: failed

**esp_err_t esp_ble_gap_set_rand_addr (esp_bd_addr_t rand_addr)**

This function sets the static Random Address and Non-Resolvable Private Address for the application.

- **rand_addr [in]** - the random address which should be setting
- **返回**
  - ESP_OK: success
  - other: failed

**esp_err_t esp_ble_gap_clear_rand_addr (void)**

This function clears the random address for the application.

- **返回**
  - ESP_OK: success
  - other: failed

**esp_err_t esp_ble_gap_config_local_privacy (bool privacy_enable)**

Enable/disable privacy on the local device.

- **privacy_enable [in]** - enable/disable privacy on remote device.
- **返回**
  - ESP_OK: success
  - other: failed

**esp_err_t esp_ble_gap_config_local_icon (uint16_t icon)**

set local gap appearance icon

- **icon [in]** - External appearance value, these values are defined by the Bluetooth SIG, please refer to [https://specificationrefs.bluetooth.com/assigned-values/Appearance%20Values.pdf](https://specificationrefs.bluetooth.com/assigned-values/Appearance%20Values.pdf)
- **返回**
  - ESP_OK: success
  - other: failed

**esp_err_t esp_ble_gap_update_whitelist (bool add_remove, esp_bd_addr_t remote_bda, esp_ble_wl_addr_type_t wl_addr_type)**

Add or remove device from white list.

- **参数**
  - **add_remove [in]** the value is true if added the ble device to the white list, and false remove to the white list.
  - **remote_bda [in]** the remote device address add/remove from the white list.
  - **wl_addr_type [in]** whitelist address type
- **返回**
  - ESP_OK: success
  - other: failed

**esp_err_t esp_ble_gap_clear_whitelist (void)**

Clear all white list.

- **返回**
Chapter 2. API

• ESP_OK: success
• other: failed

`esp_err_t esp_ble_gap_get_whitelist_size(uint16_t*length)`

Get the whitelist size in the controller.

参数 `length` - [out] the white list length.

返回
• ESP_OK: success
• other: failed

`esp_err_t esp_ble_gap_set_prefer_conn_params(esp_bd_addr_t bd_addr, uint16_t min_conn_int, uint16_t max_conn_int, uint16_t slave_latency, uint16_t supervision_tout)`

This function is called to set the preferred connection parameters when default connection parameter is not desired before connecting. This API can only be used in the master role.

参数
• `bd_addr` - [in] BD address of the peripheral
• `min_conn_int` - [in] minimum preferred connection interval
• `max_conn_int` - [in] maximum preferred connection interval
• `slave_latency` - [in] preferred slave latency
• `supervision_tout` - [in] preferred supervision timeout

返回
• ESP_OK: success
• other: failed

`esp_err_t esp_ble_gap_set_device_name(const char*name)`

Set device name to the local device.

参数 `name` - [in] - device name.

返回
• ESP_OK: success
• other: failed

`esp_err_t esp_ble_gap_get_local_used_addr(esp_bd_addr_t local_used_addr, uint8_t*addr_type)`

This function is called to get local used address and address type. `uint8_t *esp_bt_dev_get_address(void)` get the public address.

参数
• `local_used_addr` - [in] - current local used ble address (six bytes)
• `addr_type` - [in] - ble address type

返回 - ESP_OK: success
• other : failed

`uint8_t *esp_ble_resolve_adv_data(uint8_t*adv_data, uint8_t*length)`

This function is called to get ADV data for a specific type.

参数
• `adv_data` - [in] - pointer of ADV data which to be resolved
• `type` - [in] - finding ADV data type
• `length` - [out] - return the length of ADV data not including type

返回 pointer of ADV data

`esp_err_t esp_ble_gap_config_adv_data_raw(uint8_t*raw_data, uint32_t raw_data_len)`

This function is called to set raw advertising data. User need to fill ADV data by self.

参数
• `raw_data` - [in] : raw advertising data
• `raw_data_len` - [in] : raw advertising data length, less than 31 bytes

返回
• ESP_OK: success
• other: failed
**Chapter 2. API 参考**

`esp_err_t esp_ble_gap_config_scan_rsp_data_raw` *(uint8_t *raw_data, uint32_t raw_data_len)*

This function is called to set raw scan response data. User need to fill scan response data by self.

- **raw_data** - [in]: raw scan response data
- **raw_data_len** - [in]: raw scan response data length, less than 31 bytes

**返回**
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_read_rssi` *(esp_bd_addr_t remote_addr)*

This function is called to read the RSSI of remote device. The address of link policy results are returned in the gap callback function with ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT event.

- **remote_addr** - [in]: The remote connection device address.
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_add_duplicate_scan_exceptional_device` *(esp_ble_duplicate_exceptional_info_type_t type, esp_duplicate_info_t device_info)*

This function is called to add a device info into the duplicate scan exceptional list.

- **type** - [in] device info type, it is defined in esp_ble_duplicate_exceptional_info_type_t when type is MESH_BEACON_TYPE, MESH_PROV_SRV_ADV or MESH_PROXY_SRV_ADV, device_info is invalid.
- **device_info** - [in] the device information.
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_remove_duplicate_scan_exceptional_device` *(esp_ble_duplicate_exceptional_info_type_t type, esp_duplicate_info_t device_info)*

This function is called to remove a device info from the duplicate scan exceptional list.

- **type** - [in] device info type, it is defined in esp_ble_duplicate_exceptional_info_type_t when type is MESH_BEACON_TYPE, MESH_PROV_SRV_ADV or MESH_PROXY_SRV_ADV, device_info is invalid.
- **device_info** - [in] the device information.
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_clean_duplicate_scan_exceptional_list` *(esp_duplicate_scan_exceptional_list_type_t list_type)*

This function is called to clean the duplicate scan exceptional list. This API will delete all device information in the duplicate scan exceptional list.

- **list_type** - [in] duplicate scan exceptional list type, the value can be one or more of esp_duplicate_scan_exceptional_list_type_t.
- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gap_set_security_param` *(esp_ble_sm_param_t param_type, void *value, uint8_t len)*
Set a GAP security parameter value. Overrides the default value.

Secure connection is highly recommended to avoid some major vulnerabilities like 'Impersonation in the Pin Pairing Protocol' (CVE-2020-26555) and 'Authentication of the LE Legacy Pairing Protocol'.

To accept only 'secure connection mode', it is necessary do as follows:

1. Set bit `ESP_LE_AUTH_REQ_SC_ONLY` (`param_type` is `ESP_BLE_SM_AUTHEN_REQ_MODE`), bit `ESP_LE_AUTH_BOND` and bit `ESP_LE_AUTH_REQ_MITM` is optional as required.

2. Set to `ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_ENABLE` (`param_type` is `ESP_BLE_SM_ONLY_ACCEPT_SPECIFIED_SEC_AUTH`).

**参数**

- **param_type** [in]: the type of the param which to be set
- **value** [in]: the param value
- **len** [in]: the length of the param value

**返回**

- ESP_OK: success
- other: failed

**esp_err_t esp_ble_gap_security_rsp(esp_bd_addr_t bd_addr, bool accept)**

Grant security request access.

**参数**

- **bd_addr** [in]: BD address of the peer
- **accept** [in]: accept the security request or not

**返回**

- ESP_OK: success
- other: failed

**esp_err_t esp_ble_set_encryption(esp_bd_addr_t bd_addr, esp_ble_sec_act_t sec_act)**

Set a gap parameter value. Use this function to change the default GAP parameter values.

**参数**

- **bd_addr** [in]: the address of the peer device need to encryption
- **sec_act** [in]: This is the security action to indicate what kind of BLE security level is required for the BLE link if the BLE is supported

**返回**

- ESP_OK: success
- other: failed

**esp_err_t esp_ble_passkey_reply(esp_bd_addr_t bd_addr, bool accept, uint32_t passkey)**

Reply the key value to the peer device in the legacy connection stage.

**参数**

- **bd_addr** [in]: BD address of the peer
- **accept** [in]: passkey entry successful or declined.
- **passkey** [in]: passkey value, must be a 6 digit number, can be lead by 0.

**返回**

- ESP_OK: success
- other: failed

**esp_err_t esp_ble_confirm_reply(esp_bd_addr_t bd_addr, bool accept)**

Reply the confirm value to the peer device in the secure connection stage.

**参数**

- **bd_addr** [in]: BD address of the peer
• accept  - [in] : numbers to compare are the same or different.

返回 - ESP_OK : success
• other : failed

```c
esp_err_t esp_ble_remove_bond_device (esp_bd_addr_t bd_addr)
```

Removes a device from the security database list of peer device. It manages unpairing event while connected.

参数 bd_addr - [in] : BD address of the peer device
返回 - ESP_OK : success
• other : failed

```c
int esp_ble_get_bond_device_num (void)
```

Get the device number from the security database list of peer device. It will return the device bonded number immediately.

返回 - >= 0 : bonded devices number.
• ESP_FAIL : failed

```c
esp_err_t esp_ble_get_bond_device_list (int *dev_num, esp_ble_bond_dev_t *dev_list)
```

Get the device from the security database list of peer device. It will return the device bonded information immediately.

参数
• dev_num - [inout] Indicate the dev_list array(buffer) size as input. If dev_num is large enough, it means the actual number as output. Suggest that dev_num value equal to esp_ble_get_bond_device_num().
• dev_list - [out] an array(buffer) of esp_ble_bond_dev_t type. Use for storing the bonded devices address. The dev_list should be allocated by who call this API.

返回 - ESP_OK : success
• other : failed

```c
esp_err_t esp_ble_oob_req_reply (esp_bd_addr_t bd_addr, uint8_t*TK, uint8_t len)
```

This function is called to provide the OOB data for SMP in response to ESP_GAP_BLE_OOB_REQ_EVT.

参数
• bd_addr - [in] BD address of the peer device.
• TK - [in] Temporary Key value, the TK value shall be a 128-bit random number
• len - [in] length of temporary key, should always be 128-bit

返回 - ESP_OK : success
• other : failed

```c
esp_err_t esp_ble_gap_disconnect (esp_bd_addr_t remote_device)
```

This function is to disconnect the physical connection of the peer device gattc may have multiple virtual GATT server connections when multiple app_id registered. esp_ble_gattc_close (esp_ble_if_t gattc_if, uint16_t conn_id) only close one virtual GATT server connection. if there exist other virtual GATT server connections, it does not disconnect the physical connection. esp_ble_gap_disconnect(esp_bd_addr_t remote_device) disconnect the physical connection directly.

参数 remote_device - [in] : BD address of the peer device
返回 - ESP_OK : success
• other : failed

```c
esp_err_t esp_ble_get_current_conn_params (esp_bd_addr_t bd_addr, esp_gap_conn_params_t *conn_params)
```

This function is called to read the connection parameters information of the device.

参数
• bd_addr - [in] BD address of the peer device.
• conn_params - [out] the connection parameters information

返回 - ESP_OK : success
• other : failed
esp_err_t esp_gap_ble_set_channels(esp_gap_ble_channels channels)
BLE set channels.
参数 channels [in]: The n th such field (in the range 0 to 36) contains the value for the link layer channel index n. 0 means channel n is bad. 1 means channel n is unknown. The most significant bits are reserved and shall be set to 0. At least one channel shall be marked as unknown.
返回 - ESP_OK : success
  • ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
  • other : failed

esp_err_t esp_gap_ble_set_authorization(esp_bd_addr_t bd_addr, bool authorize)
This function is called to authorized a link after Authentication(MITM protection)
参数
  • bd_addr –[in] BD address of the peer device.
  • authorize –[out] Authorized the link or not.
返回 - ESP_OK : success
  • other : failed

esp_err_t esp_ble_gap_read_phy(esp_bd_addr_t bd_addr)
This function is used to read the current transmitter PHY and receiver PHY on the connection identified by remote address.
参数 bd_addr –[in]: BD address of the peer device.
返回 - ESP_OK : success
  • other : failed

esp_err_t esp_ble_gap_set_preferred_default_phy(esp_ble_gap_phy_mask_t tx_phy_mask, esp_ble_gap_phy_mask_t rx_phy_mask)
This function is used to allows the Host to specify its preferred values for the transmitter PHY and receiver PHY to be used for all subsequent connections over the LE transport.
参数
  • tx_phy_mask –[in]: indicates the transmitter PHYs that the Host prefers the Controller to use
  • rx_phy_mask –[in]: indicates the receiver PHYs that the Host prefers the Controller to use
返回 - ESP_OK : success
  • other : failed

esp_err_t esp_ble_gap_set_preferred_phy(esp_bd_addr_t bd_addr, esp_ble_gap_all_phys_t all_phys_mask, esp_ble_gap_phy_mask_t tx_phy_mask, esp_ble_gap_phy_mask_t rx_phy_mask, esp_ble_gap_prefer_phy_options_t phy_options)
This function is used to set the PHY preferences for the connection identified by the remote address. The Controller might not be able to make the change (e.g. because the peer does not support the requested PHY) or may decide that the current PHY is preferable.
参数
  • bd_addr –[in]: remote address
  • all_phys_mask –[in]: a bit field that allows the Host to specify
  • tx_phy_mask –[in]: a bit field that indicates the transmitter PHYs that the Host prefers the Controller to use
  • rx_phy_mask –[in]: a bit field that indicates the receiver PHYs that the Host prefers the Controller to use
  • phy_options –[in]: a bit field that allows the Host to specify options for PHYs
返回 - ESP_OK : success
  • other : failed

esp_err_t esp_ble_gap_ext_adv_set_rand_addr(uint8_t instance, esp_bd_addr_t rand_addr)
This function is used by the Host to set the random device address specified by the Random_Address parameter.
**Chapter 2. API 参考**

**参数**
- `instance` - [in]: Used to identify an advertising set
- `rand_addr` - [in]: Random Device Address

**返回**
- ESP_OK : success
- other : failed

`esp_err_t esp_ble_gap_ext_adv_set_params (uint8_t instance, const esp_ble_gap_ext_adv_params_t *params)`

This function is used by the Host to set the advertising parameters.

**参数**
- `instance` - [in]: identifies the advertising set whose parameters are being configured.
- `params` - [in]: advertising parameters

**返回**
- ESP_OK : success
- other : failed

`esp_err_t esp_ble_gap_config_ext_adv_data_raw (uint8_t instance, uint16_t length, const uint8_t *data)`

This function is used to set the data used in advertising PDUs that have a data field.

**参数**
- `instance` - [in]: identifies the advertising set whose data are being configured.
- `length` - [in]: data length
- `data` - [in]: data information

**返回**
- ESP_OK : success
- other : failed

`esp_err_t esp_ble_gap_config_ext_scan_rsp_data_raw (uint8_t instance, uint16_t length, const uint8_t *scan_rsp_data)`

This function is used to provide scan response data used in scanning response PDUs.

**参数**
- `instance` - [in]: identifies the advertising set whose response data are being configured.
- `length` - [in]: response data length
- `scan_rsp_data` - [in]: response data information

**返回**
- ESP_OK : success
- other : failed

`esp_err_t esp_ble_gap_ext_adv_start (uint8_t num_adv, const esp_ble_gap_ext_adv_t *ext_adv)`

This function is used to request the Controller to enable one or more advertising sets using the advertising sets identified by the instance parameter.

**参数**
- `num_adv` - [in]: Number of advertising sets to enable or disable
- `ext_adv` - [in]: adv parameters

**返回**
- ESP_OK : success
- other : failed

`esp_err_t esp_ble_gap_ext_adv_stop (uint8_t num_adv, const uint8_t *ext_adv_inst)`

This function is used to request the Controller to disable one or more advertising sets using the advertising sets identified by the instance parameter.

**参数**
- `num_adv` - [in]: Number of advertising sets to enable or disable
- `ext_adv_inst` - [in]: ext adv instance

**返回**
- ESP_OK : success
- other : failed

`esp_err_t esp_ble_gap_ext_adv_set_remove (uint8_t instance)`

This function is used to remove an advertising set from the Controller.

**参数**
- `instance` - [in]: Used to identify an advertising set
Chapter 2. API 参考

返回 - ESP_OK : success
  • other : failed

`esp_err_t esp_ble_gap_ext_adv_set_clear(void)`

This function is used to remove all existing advertising sets from the Controller.

返回 - ESP_OK : success
  • other : failed

`esp_err_t esp_ble_gap_periodic_adv_set_params(uint8_t instance, const esp_ble_gap_periodic_adv_params_t *params)`

This function is used by the Host to set the parameters for periodic advertising.

参数
  • instance [in] : identifies the advertising set whose periodic advertising parameters are being configured.
  • params [in] : periodic adv parameters

返回 - ESP_OK : success
  • other : failed

`esp_err_t esp_ble_gap_config_periodic_adv_data_raw(uint8_t instance, uint16_t length, const uint8_t *data)`

This function is used to set the data used in periodic advertising PDUs.

参数
  • instance [in] : identifies the advertising set whose periodic advertising parameters are being configured.
  • length [in] : the length of periodic data
  • data [in] : periodic data information

返回 - ESP_OK : success
  • other : failed

`esp_err_t esp_ble_gap_periodic_adv_start(uint8_t instance)`

This function is used to request the Controller to enable the periodic advertising for the advertising set specified.

参数 instance [in] : Used to identify an advertising set

返回 - ESP_OK : success
  • other : failed

`esp_err_t esp_ble_gap_periodic_adv_stop(uint8_t instance)`

This function is used to request the Controller to disable the periodic advertising for the advertising set specified.

参数 instance [in] : Used to identify an advertising set

返回 - ESP_OK : success
  • other : failed

`esp_err_t esp_ble_gap_set_ext_scan_params(const esp_ext_scan_params_t *params)`

This function is used to set the extended scan parameters to be used on the advertising channels.

参数 params [in] : scan parameters

返回 - ESP_OK : success
  • other : failed

`esp_err_t esp_ble_gap_start_ext_scan(uint32_t duration, uint16_t period)`

This function is used to enable scanning.

参数
  • duration [in] : Scan duration
  • period [in] : Time interval from when the Controller started its last Scan Duration until it begins the subsequent Scan Duration.

返回 - ESP_OK : success
  • other : failed
**esp_err_t esp_ble_gap_stop_ext_scan (void)**

This function is used to disable scanning.

- **ESP_OK**: success
- **other**: failed

**esp_err_t esp_ble_gap_periodic_adv_create_sync (const esp_ble_gap_periodic_adv_sync_params_t *params)**

This function is used to synchronize with periodic advertising from an advertiser and begin receiving periodic advertising packets.

- 参数 **params** - [in]: sync parameters
- 返回 - **ESP_OK**: success
- **other**: failed

**esp_err_t esp_ble_gap_periodic_adv_sync_cancel (void)**

This function is used to cancel the LE_Periodic_Advertising_Create_Sync command while it is pending.

- 返回 - **ESP_OK**: success
- **other**: failed

**esp_err_t esp_ble_gap_periodic_adv_sync_terminate (uint16_t sync_handle)**

This function is used to stop reception of the periodic advertising identified by the Sync Handle parameter.

- 参数 **sync_handle** - [in]: identify the periodic advertiser
- 返回 - **ESP_OK**: success
- **other**: failed

**esp_err_t esp_ble_gap_periodic_adv_add_dev_to_list (esp_ble_addr_type_t addr_type, esp_bd_addr_t addr, uint8_t sid)**

This function is used to add a single device to the Periodic Advertiser list stored in the Controller.

- 参数
  - **addr_type** - [in]: address type
  - **addr** - [in]: Device Address
  - **sid** - [in]: Advertising SID subfield in the ADI field used to identify the Periodic Advertising
- 返回 - **ESP_OK**: success
- **other**: failed

**esp_err_t esp_ble_gap_periodic_adv_remove_dev_from_list (esp_ble_addr_type_t addr_type, esp_bd_addr_t addr, uint8_t sid)**

This function is used to remove one device from the list of Periodic Advertisers stored in the Controller. Removals from the Periodic Advertisers List take effect immediately.

- 参数
  - **addr_type** - [in]: address type
  - **addr** - [in]: Device Address
  - **sid** - [in]: Advertising SID subfield in the ADI field used to identify the Periodic Advertising
- 返回 - **ESP_OK**: success
- **other**: failed

**esp_err_t esp_ble_gap_periodic_adv_clear_dev (void)**

This function is used to remove all devices from the list of Periodic Advertisers in the Controller.

- 返回 - **ESP_OK**: success
- **other**: failed
```c
esp_err_t esp_ble_gap_prefer_ext_connect_params_set(esp_bd_addr_t addr,
        esp_ble_gap_phy_mask_t phy_mask,
        const esp_ble_gap_conn_params_t *phy_1m_conn_params,
        const esp_ble_gap_conn_params_t *phy_2m_conn_params,
        const esp_ble_gap_conn_params_t *phy_coded_conn_params)
```

This function is used to set aux connection parameters.

**Parameters**
- `addr` - [in]: device address
- `phy_mask` - [in]: indicates the PHY(s) on which the advertising packets should be received on the primary advertising channel and the PHYs for which connection parameters have been specified.
- `phy_1m_conn_params` - [in]: Scan connectable advertisements on the LE 1M PHY. Connection parameters for the LE 1M PHY are provided.
- `phy_2m_conn_params` - [in]: Connection parameters for the LE 2M PHY are provided.
- `phy_coded_conn_params` - [in]: Scan connectable advertisements on the LE Coded PHY. Connection parameters for the LE Coded PHY are provided.

**Return**
- ESP_OK: success
- other: failed

**Unions**

```c
union esp_ble_key_value_t
#include <esp_gap_ble_api.h> union type of the security key value
```

**Public Members**

```c
esp_ble_penc_keys_t penc_key
received peer encryption key
```

```c
esp_ble_pcsrk_keys_t pcsrk_key
received peer device SRK
```

```c
esp_ble_pid_keys_t pid_key
peer device ID key
```

```c
esp_ble_lenc_keys_t lenc_key
local encryption reproduction keys LTK = = d1(ER,DIV,0)
```

```c
esp_ble_lcsrk_keys lcsrk_key
local device CSRK = d1(ER,DIV,1)
```

**Unions**

```c
union esp_ble_sec_t
#include <esp_gap_ble_api.h> union associated with ble security
```

**Public Members**

```c
Espressif Systems 253 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
```
Chapter 2. API

esp_ble_sec_key_notif_t keynotif
passkey notification

esp_ble_sec_req_t ble_req
BLE SMP related request

esp_ble_key_t ble_key
BLE SMP keys used when pairing

esp_ble_local_id_keys_t ble_id_keys
BLE IR event

esp_ble_auth_cmpl_t auth_cmpl
Authentication complete indication.

union esp_ble_gap_cb_param_t
#include <esp_gap_ble_api.h> Gap callback parameters union.

Public Members

struct esp_ble_gap_cb_param_t::ble_adv_data_cmpl_evt_param adv_data_cmpl
Event parameter of ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_scan_rsp_data_cmpl_evt_param scan_rsp_data_cmpl
Event parameter of ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_scan_param_cmpl_evt_param scan_param_cmpl
Event parameter of ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_scan_result_evt_param scan_rst
Event parameter of ESP_GAP_BLE_SCAN_RESULT_EVT

struct esp_ble_gap_cb_param_t::ble_adv_data_raw_cmpl_evt_param adv_data_raw_cmpl
Event parameter of ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_scan_rsp_data_raw_cmpl_evt_param scan_rsp_data_raw_cmpl
Event parameter of ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_adv_start_cmpl_evt_param adv_start_cmpl
Event parameter of ESP_GAP_BLE_ADV_START_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_scan_start_cmpl_evt_param scan_start_cmpl
Event parameter of ESP_GAP_BLE_SCAN_START_COMPLETE_EVT

esp_ble_sec_t ble_security
ble gap security union type
struct esp_ble_gap_cb_param_t::ble_scan_stop_cmpl_evt_param scan_stop_cmpl
Event parameter of ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_adv_stop_cmpl_evt_param adv_stop_cmpl
Event parameter of ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_set_rand_cmpl_evt_param set_rand_addr_cmpl
Event parameter of ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT

struct esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param update_conn_params
Event parameter of ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT

struct esp_ble_gap_cb_param_t::ble_pkt_data_length_cmpl_evt_param pkt_data_length_cmpl
Event parameter of ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_local_privacy_cmpl_evt_param local_privacy_cmpl
Event parameter of ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_remove_bond_dev_cmpl_evt_param remove_bond_dev_cmpl
Event parameter of ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_clear_bond_dev_cmpl_evt_param clear_bond_dev_cmpl
Event parameter of ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_get_bond_dev_cmpl_evt_param get_bond_dev_cmpl
Event parameter of ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_read_rssi_cmpl_evt_param read_rssi_cmpl
Event parameter of ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_update_whitelist_cmpl_evt_param update_whitelist_cmpl
Event parameter of ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_update_duplicate_exceptional_list_cmpl_evt_param update_duplicate_exceptional_list_cmpl
Event parameter of ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_set_channels_evt_param ble_set_channels
Event parameter of ESP_GAP_BLE_SET_CHANNELS_EVT

struct esp_ble_gap_cb_param_t::ble_read_phy_cmpl_evt_param read_phy
Event parameter of ESP_GAP_BLE_READ_PHY_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_set_perf_def_phy_cmpl_evt_param set_perf_def_phy
Event parameter of ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_set_perf_phy_cmpl_evt_param set_perf_phy
Event parameter of ESP_GAP_BLE_SET_PREFERRED_PHY_COMPLETE_EVT
struct esp_ble_gap_cb_param_t::ble_ext_adv_set_rand_addr_cmpl_evt_param
struct ext_adv_set_rand_addr
Event parameter of ESP_GAP_BLE_EXT_ADV_SET_RAND_ADDR_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_set_params_cmpl_evt_param
struct ext_adv_set_params
Event parameter of ESP_GAP_BLE_EXT_ADV_SET_PARAMS_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_data_set_cmpl_evt_param
struct ext_adv_data_set
Event parameter of ESP_GAP_BLE_EXT_ADV_DATA_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_scan_rsp_set_cmpl_evt_param
struct scan_rsp_set
Event parameter of ESP_GAP_BLE_EXT_SCAN_RSP_DATA_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_start_cmpl_evt_param
struct ext_adv_start
Event parameter of ESP_GAP_BLE_EXT_ADV_START_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_stop_cmpl_evt_param
struct ext_adv_stop
Event parameter of ESP_GAP_BLE_EXT_ADV_STOP_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_set_remove_cmpl_evt_param
struct ext_adv_remove
Event parameter of ESP_GAP_BLE_EXT_ADV_SET_REMOVE_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_set_clear_cmpl_evt_param
struct ext_adv_clear
Event parameter of ESP_GAP_BLE_EXT_ADV_SET_CLEAR_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_periodic_adv_set_params_cmpl_param
struct peroid_adv_set_params
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SET_PARAMS_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_periodic_adv_data_set_cmpl_param
struct period_adv_data_set
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_DATA_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_periodic_adv_start_cmpl_param
struct period_adv_start
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_START_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_periodic_adv_stop_cmpl_param
struct period_adv_stop
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_STOP_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_period_adv_create_sync_cmpl_param
struct period_adv_create_sync
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_CREATE_SYNC_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_period_adv_sync_cancel_cmpl_param
struct period_adv_sync_cancel
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_CANCEL_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_period_adv_sync_terminate_cmpl_param
struct period_adv_sync_term
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_TERMINATE_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_period_adv_add_dev_cmpl_param
struct period_adv_add_dev
Event parameter of ESP_GAP_BLE_PERIODIC_ADV_ADD_DEV_COMPLETE_EVT
struct esp_ble_gap_cb_param_t::ble_period_adv_remove_dev_cmpl_param period_adv_remove_dev
    Event parameter of ESP_GAP_BLE_PERIODIC_ADV_REMOVE_DEV_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_period_adv_clear_dev_cmpl_param period_adv_clear_dev
    Event parameter of ESP_GAP_BLE_PERIODIC_ADV_CLEAR_DEV_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_set_ext_scan_params_cmpl_param set_ext_scan_params
    Event parameter of ESP_GAP_BLE_SET_EXT_SCAN_PARAMS_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_scan_start_cmpl_param ext_scan_start
    Event parameter of ESP_GAP_BLE_EXT_SCAN_START_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_scan_stop_cmpl_param ext_scan_stop
    Event parameter of ESP_GAP_BLE_EXT_SCAN_STOP_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_conn_params_set_cmpl_param ext_conn_params_set
    Event parameter of ESP_GAP_BLE_PREFER_EXT_CONN_PARAMS_SET_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_adv_terminate_param adv_terminate
    Event parameter of ESP_GAP_BLE_ADV_TERMINATED_EVT

struct esp_ble_gap_cb_param_t::ble_scan_req_received_param scan_req_received
    Event parameter of ESP_GAP_BLE_SCAN_REQ_RECEIVED_EVT

struct esp_ble_gap_cb_param_t::ble_channel_sel_alg_param channel_sel_alg
    Event parameter of ESP_GAP_BLE_CHANNEL_SELECT_ALGORITHM_EVT

struct esp_ble_gap_cb_param_t::ble_periodic_adv_sync_lost_param periodic_adv_sync_lost
    Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_LOST_EVT

struct esp_ble_gap_cb_param_t::ble_periodic_adv_sync_estab_param periodic_adv_sync_estab
    Event parameter of ESP_GAP_BLE_PERIODIC_ADV_SYNC_ESTAB_EVT

struct esp_ble_gap_cb_param_t::ble_phy_update_cmpl_param phy_update
    Event parameter of ESP_GAP_BLE_PHY_UPDATE_COMPLETE_EVT

struct esp_ble_gap_cb_param_t::ble_ext_adv_report_param ext_adv_report
    Event parameter of ESP_GAP_BLE_EXT_ADV_REPORT_EVT

struct esp_ble_gap_cb_param_t::ble_periodic_adv_report_param period_adv_report
    Event parameter of ESP_GAP_BLE_PERIODIC_ADV_REPORT_EVT

struct ble_adv_data_cmpl_evt_param
    #include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT.

**Public Members**
### Chapter 2. API Reference

*esp.bt_status_t* **status**  
Indicate the set advertising data operation success status

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT.
```

**Public Members**

*esp.bt_status_t* **status**  
Indicate the set raw advertising data operation success status

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_RAW_COMPLETE_EVT.
```

**Public Members**

*esp.bt_status_t* **status**  
Indicate advertising start operation success status

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_START_COMPLETE_EVT.
```

**Public Members**

*esp.bt_status_t* **status**  
Indicate adv stop operation success status

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT.
```

**Public Members**

uint8_t **status**  
Indicate adv terminate status

uint8_t **adv_instance**  
extend advertising handle

uint16_t **conn_idx**  
connection index

uint8_t **completed_event**  
the number of completed extend advertising events

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_TERMINATED_EVT.
```

**Public Members**

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_CHANNEL_SELECT_ALGORITHM_EVT.
```
Public Members

uint16_t **conn_handle**
connection handle

uint8_t **channel_sel_alg**
channel selection algorithm

struct **ble_clear_bond_dev_cmpl_evt_param**
#include <esp_gap_ble_api.h> ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT.

Public Members

*esp_bt_status_t* **status**
Indicate the clear bond device operations success status

struct **ble_ext_adv_data_set_cmpl_evt_param**
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_DATA_SET_COMPLETE_EVT.

Public Members

*esp_bt_status_t* **status**
Indicate extend advertising data set status

struct **ble_ext_adv_report_param**
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_REPORT_EVT.

Public Members

*esp_ble_gap_ext_adv_report_t* **params**
extend advertising report parameters

struct **ble_ext_adv_scan_rsp_set_cmpl_evt_param**
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_SCAN_RSP_DATA_SET_COMPLETE_EVT.

Public Members

*esp_bt_status_t* **status**
Indicate extend advertising scan response data set status

struct **ble_ext_adv_set_clear_cmpl_evt_param**
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_CLEAR_COMPLETE_EVT.
Public Members

```c
struct ble_ext_adv_set_params_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_PARAMS_COMPLETE_EVT.
```

Public Members

```c
struct ble_ext_adv_set_rand_addr_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_RAND_ADDR_COMPLETE_EVT.
```

Public Members

```c
struct ble_ext_adv_set_remove_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_SET_REMOVE_COMPLETE_EVT.
```

Public Members

```c
struct ble_ext_adv_start_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_START_COMPLETE_EVT.
```

Public Members

```c
struct ble_ext_adv_stop_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_ADV_STOP_COMPLETE_EVT.
```
struct ble_ext_conn_params_set_cmpl_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PREFER_EXT_CONN_PARAMS_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t status
Indicate extend connection parameters set status

struct ble_ext_scan_start_cmpl_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_SCAN_START_COMPLETE_EVT.

Public Members

esp_bt_status_t status
Indicate extend advertising start status

struct ble_ext_scan_stop_cmpl_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_EXT_SCAN_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t status
Indicate extend advertising stop status

struct ble_get_bond_dev_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t status
Indicate the get bond device operation success status

uint8_t dev_num
Indicate the get number device in the bond list

esp_ble_bond_dev_t *bond_dev
the pointer to the bond device Structure

struct ble_local_privacy_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT.

Public Members

esp_bt_status_t status
Indicate the set local privacy operation success status
struct `ble_period_adv_add_dev_cmpl_param`
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_ADD_DEV_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising device list add status

struct `ble_period_adv_clear_dev_cmpl_param`
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_CLEAR_DEV_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising device list clean status

struct `ble_period_adv_create_sync_cmpl_param`
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_CREATE_SYNC_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising create sync status

struct `ble_period_adv_remove_dev_cmpl_param`
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_REMOVE_DEV_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising device list remove status

struct `ble_period_adv_sync_cancel_cmpl_param`
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SYNC_CANCEL_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising sync cancel status

struct `ble_period_adv_sync_terminate_cmpl_param`
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SYNC_TERMINATE_COMPLETE_EVT.
Public Members

`esp_bt_status_t status`
Indicate periodic advertising sync terminate status

struct `ble_periodic_adv_data_set_cmpl_param`
#include `<esp_gap_ble_api.h>` ESP_GAP_BLE_PERIODIC_ADV_DATA_SET_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising data set status

struct `ble_periodic_adv_report_param`
#include `<esp_gap_ble_api.h>` ESP_GAP_BLE_PERIODIC_ADV_REPORT_EVT.

Public Members

`esp_ble_gap_periodic_adv_report_t params`
periodic advertising report parameters

struct `ble_periodic_adv_set_params_cmpl_param`
#include `<esp_gap_ble_api.h>` ESP_GAP_BLE_PERIODIC_ADV_SET_PARAMS_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising parameters set status

struct `ble_periodic_adv_start_cmpl_param`
#include `<esp_gap_ble_api.h>` ESP_GAP_BLE_PERIODIC_ADV_START_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising start status

struct `ble_periodic_adv_stop_cmpl_param`
#include `<esp_gap_ble_api.h>` ESP_GAP_BLE_PERIODIC_ADV_STOP_COMPLETE_EVT.

Public Members

`esp_bt_status_t status`
Indicate periodic advertising stop status
struct ble_periodic_adv_sync_estab_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SYNC_ESTAB_EVT.

Public Members

uint8_t status
periodic advertising sync status

uint16_t sync_handle
periodic advertising sync handle

uint8_t sid
periodic advertising sid

esp_ble_addr_type_t adv_addr_type
periodic advertising address type

esp_bd_addr_t adv_addr
periodic advertising address

esp_ble_gap Phy_t adv_phy
periodic advertising phy type

uint16_t period_adv_interval
periodic advertising interval

uint8_t adv_clk_accuracy
periodic advertising clock accuracy

struct ble_periodic_adv_sync_lost_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PERIODIC_ADV_SYNC_LOST_EVT.

Public Members

uint16_t sync_handle
sync handle

struct ble_phy_update_cmpl_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_PHY_UPDATE_COMPLETE_EVT.

Public Members

esp_bt_status_t status
phy update status
Chapter 2. API

\begin{verbatim}
    esp_bd_addr_t bda
        address

    esp_ble_gap_phy_t tx_phy
        tx phy type

    esp_ble_gap_phy_t rx_phy
        rx phy type

struct ble_pkt_data_length_cmpl_evt_param
    #include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT.

Public Members

    esp_bt_status_t status
        Indicate the set pkt data length operation success status

    esp_ble_pkt_data_length_params_t params
        pkt data length value

struct ble_read_phy_cmpl_evt_param
    #include <esp_gap_ble_api.h> ESP_GAP_BLE_READ_PHY_COMPLETE_EVT.

Public Members

    esp_bt_status_t status
        read phy complete status

    esp_bd_addr_t bda
        read phy address

    esp_ble_gap_phy_t tx_phy
        tx phy type

    esp_ble_gap_phy_t rx_phy
        rx phy type

struct ble_read_rssi_cmpl_evt_param
    #include <esp_gap_ble_api.h> ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT.

Public Members

    esp_bt_status_t status
        Indicate the read adv tx power operation success status
\end{verbatim}
int8_t rssi
The ble remote device rssi value, the range is from -127 to 20, the unit is dbm, if the RSSI cannot be read, the RSSI metric shall be set to 127.

esp_bd_addr_t remote_addr
The remote device address

struct ble_remove_bond_dev_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t status
Indicate the remove bond device operation success status

esp_bd_addr_t bd_addr
The device address which has been remove from the bond list

struct ble_scan_param_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT.

Public Members

esp bt status t status
Indicate the set scan param operation success status

struct ble_scan_req_received_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_REQ_RECEIVED_EVT.

Public Members

uint8_t adv_instance
extend advertising handle

esp_ble_addr_type_t scan_addr_type
scanner address type

esp bd addr t scan addr
scanner address

struct ble_scan_result_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RESULT_EVT.
Public Members

`esp_gap_search_evt_t search_evt`
Search event type

`esp_bd_addr_t bda`
Bluetooth device address which has been searched

`esp_bt_dev_type_t dev_type`
Device type

`esp_ble_addr_type_t ble_addr_type`
Ble device address type

`esp_ble_evt_type_t ble_evt_type`
Ble scan result event type

`int rssi`
Searched device’s RSSI

`uint8_t ble_adv[ESP_BLE_ADV_DATA_LEN_MAX + ESP_BLE_SCAN_RSP_DATA_LEN_MAX]`
Received EIR

`int flag`
Advertising data flag bit

`int num_resps`
Scan result number

`uint8_t adv_data_len`
Adv data length

`uint8_t scan_rsp_len`
Scan response length

`uint32_t num_dis`
The number of discard packets

`struct ble_scan_rsp_data_cmpl_evt_param`

```
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT.
```

Public Members

`esp_bt_status_t status`
Indicate the set scan response data operation success status

`struct ble_scan_rsp_data_raw_cmpl_evt_param`

```
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT.
```
Public Members

`esp_bt_status_t status`
Indicates the set raw advertising data operation success status

```c
#include <esp_gap_ble_api.h>
ESP_GAP_BLE_SCAN_START_COMPLETE_EVT.
```

Public Members

`esp_bt_status_t status`
Indicate scan start operation success status

```c
#include <esp_gap_ble_api.h>
ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT.
```

Public Members

`esp_bt_status_t status`
Indicate scan stop operation success status

```c
#include <esp_gap_ble_api.h>
ESP_GAP_BLE_SET_CHANNELS_EVT.
```

Public Members

`esp_bt_status_t stat`
BLE set channel status

```c
#include <esp_gap_ble_api.h>
ESP_GAP_BLE_SET_EXT_SCAN_PARAMS_COMPLETE_EVT.
```

Public Members

`esp_bt_status_t status`
Indicate extend advertising parameters set status

```c
#include <esp_gap_ble_api.h>
ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT.
```

Public Members

`esp_bt_status_t status`
Indicate perf default phy set status
struct `ble_set_perf_phy_cmpl_evt_param`

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_PREFERRED_PHY_COMPLETE_EVT.
```

**Public Members**

```c
esp_bt_status_t status
```

Indicate perf phy set status

struct `ble_set_rand_cmpl_evt_param`

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT.
```

**Public Members**

```c
esp_bt_status_t status
```

Indicate set static rand address operation success status

struct `ble_update_conn_params_evt_param`

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT.
```

**Public Members**

```c
esp_bt_status_t status
```

Indicate update connection parameters success status

```c
esp_bd_addr_t bda
```

Bluetooth device address

```c
uint16_t min_int
```

Min connection interval

```c
uint16_t max_int
```

Max connection interval

```c
uint16_t latency
```

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

```c
uint16_t conn_int
```

Current connection interval

```c
uint16_t timeout
```

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec

struct `ble_update_duplicate_exceptional_list_cmpl_evt_param`

```c
#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT.
```
**Public Members**

```c
esp_bt_status_t status
```
Indicate update duplicate scan exceptional list operation success status

```c
uint8_t subcode
```
Define in `esp_bt_duplicate_exceptional_subcode_type_t`

```c
uint16_t length
```
The length of `device_info`

```c
esp_duplicate_info_t device_info
```
Device information, when subcode is `ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_CLEAN`, the value is invalid

```c
#include <esp_gap_ble_api.h>
```

# ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT

**Public Members**

```c
esp_bt_status_t status
```
Indicate the add or remove whitelist operation success status

```c
estp_ble_wl_operation_t wl_operation
```
The value is `ESP_BLE_WHITELIST_ADD` if add address to whitelist operation success, `ESP_BLE_WHITELIST_REMOVE` if remove address from the whitelist operation success

**Structures**

```c
struct esp_ble_adv_params_t
```
Advertising parameters.

**Public Members**

```c
uint16_t adv_int_min
```
Minimum advertising interval for undirected and low duty cycle directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec

```c
uint16_t adv_int_max
```
Maximum advertising interval for undirected and low duty cycle directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec Advertising max interval

```c
esp_ble_adv_type_t adv_type
```
Advertising type

```c
esp_ble_addr_type_t own_addr_type
```
Owner Bluetooth device address type
**Chapter 2. API**

- `esp_bd_addr_t peer_addr`
  Peer device bluetooth device address

- `esp_ble_addr_type_t peer_addr_type`
  Peer device bluetooth device address type, only support public address type and random address type

- `esp_ble_adv_channel_t channel_map`
  Advertising channel map

- `esp_ble_adv_filter_t adv_filter_policy`
  Advertising filter policy

```c
struct esp_ble_adv_data_t
```
Advertising data content, according to “Supplement to the Bluetooth Core Specification”.

**Public Members**

- `bool set_scan_rsp`
  Set this advertising data as scan response or not

- `bool include_name`
  Advertising data include device name or not

- `bool include_txpower`
  Advertising data include TX power

- `int min_interval`
  Advertising data show slave preferred connection min interval. The connection interval in the following manner: `connIntervalMin = Conn_INTERVAL_Min * 1.25 ms` Conn_INTERVAL_Min range: 0x0006 to 0x0C80 Value of 0xFFFF indicates no specific minimum. Values not defined above are reserved for future use.

- `int max_interval`
  Advertising data show slave preferred connection max interval. The connection interval in the following manner: `connIntervalMax = Conn_INTERVAL_Max * 1.25 ms` Conn_INTERVAL_Max range: 0x0006 to 0x0C80 Conn_INTERVAL_Max shall be equal to or greater than the Conn_INTERVAL_Min. Value of 0xFFFF indicates no specific maximum. Values not defined above are reserved for future use.

- `int appearance`
  External appearance of device

- `uint16_t manufacturer_len`
  Manufacturer data length

- `uint8_t* p_manufacturer_data`
  Manufacturer data point

- `uint16_t service_data_len`
  Service data length
```c
uint8_t *p_service_data
    Service data point

uint16_t service_uuid_len
    Service uuid length

uint8_t *p_service_uuid
    Service uuid array point

uint8_t flag
    Advertising flag of discovery mode, see BLE_ADV_DATA_FLAG detail

struct esp_ble_scan_params_t
    Ble scan parameters.

Public Members

esp_ble_scan_type_t scan_type
    Scan type

esp_ble_addr_type_t own_addr_type
    Owner address type

esp_ble_scan_filter_t scan_filter_policy
    Scan filter policy

uint16_t scan_interval
    Scan interval. This is defined as the time interval from when the Controller started its last LE scan until it begins the subsequent LE scan. Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10.24 seconds

uint16_t scan_window
    Scan window. The duration of the LE scan. LE_Scan_Window shall be less than or equal to LE_Scan_Interval Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10240 msec

esp_ble_scan_duplicate_t scan_duplicate
    The Scan_Duplicates parameter controls whether the Link Layer should filter out duplicate advertising reports (BLE_SCAN_DUPLICATE_ENABLE) to the Host, or if the Link Layer should generate advertising reports for each packet received

struct esp_gap_conn_params_t
    connection parameters information

Public Members

uint16_t interval
    connection interval
```
uint16_t latency
Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t timeout
Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80
Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct esp_ble_conn_update_params_t
Connection update parameters.

Public Members

esp_bd_addr_t bda
Bluetooth device address

uint16_t min_int
Min connection interval

uint16_t max_int
Max connection interval

uint16_t latency
Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t timeout
Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80
Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct esp_ble_pkt_data_length_params_t
BLE pkt date length keys.

Public Members

uint16_t rx_len
pkt rx data length value

uint16_t tx_len
pkt tx data length value

struct esp_ble_penc_keys_t
BLE encryption keys.

Public Members

esp_bt_octet16_t ltk
The long term key
```

``exp_bt_octet8_t rand``

  The random number

``uint16_t ediv``

  The ediv value

``uint8_t sec_level``

  The security level of the security link

``uint8_t key_size``

  The key size(7~16) of the security link

struct `esp_ble_pcsrk_keys_t`

  BLE CSRK keys.

  **Public Members**

``uint32_t counter``

  The counter

``exp.bt_octet16_t csrk``

  The csrk key

``uint8_t sec_level``

  The security level

struct `esp_ble_pid_keys_t`

  BLE pid keys.

  **Public Members**

``exp.bt_octet16_t irk``

  The irk value

``exp_ble_addr_type_t addr_type``

  The address type

``exp_bd_addr_t static_addr``

  The static address

struct `esp_ble_lenc_keys_t`

  BLE Encryption reproduction keys.

  **Public Members**
**Chapter 2. API Reference**

*esp_bt_octet16_t* **ltk**
The long term key

*uint16_t* **div**
The div value

*uint8_t* **key_size**
The key size of the security link

*uint8_t* **sec_level**
The security level of the security link

**struct esp_ble_lcsrk_keys**
BLE SRK keys.

**Public Members**

*uint32_t* **counter**
The counter value

*uint16_t* **div**
The div value

*uint8_t* **sec_level**
The security level of the security link

*esp_bt_octet16_t* **csrk**
The csrk key value

**struct esp_ble_sec_key_notif_t**
Structure associated with ESP_KEY_NOTIF_EVT.

**Public Members**

*esp_bd_addr_t* **bd_addr**
peer address

*uint32_t* **passkey**
the numeric value for comparison. If just_works, do not show this number to UI

**struct esp_ble_sec_req_t**
Structure of the security request.

**Public Members**
exp_bd_addr_t bd_addr
    peer address

struct esp_ble_bond_key_info_t
    struct type of the bond key information value

Public Members

exp_ble_key_mask_t key_mask
    the key mask to indicate which key is present

desp_ble_penc_keys_t penc_key
    received peer encryption key

desp_ble_pcsrk_keys_t pcsrk_key
    received peer device SRK

desp_ble_pid_keys_t pid_key
    peer device ID key

struct esp_ble_bond_dev_t
    struct type of the bond device value

Public Members

exp_bd_addr_t bd_addr
    peer address

desp_ble_bond_key_info_t bond_key
    the bond key information

struct esp_ble_key_t
    union type of the security key value

Public Members

exp_bd_addr_t bd_addr
    peer address

desp_ble_key_type_t key_type
    key type of the security link

desp_ble_key_value_t p_key_value
    the pointer to the key value

struct esp_ble_local_id_keys_t
    structure type of the ble local id keys value
**Public Members**

`esp_bt_octet16_t ir`
the 16 bits of the ir value

`esp_bt_octet16_t irk`
the 16 bits of the ir key value

`esp_bt_octet16_t dhk`
the 16 bits of the dh key value

`struct esp_ble_auth_cmpl_t`
Structure associated with ESP_AUTH_CMPL_EVT.

**Public Members**

`esp_bd_addr_t bd_addr`
BD address peer device.

`bool key_present`
Valid link key value in key element

`esp_link_key key`
Link key associated with peer device.

`uint8_t key_type`
The type of Link Key

`bool success`
TRUE of authentication succeeded, FALSE if failed.

`uint8_t fail_reason`
The HCI reason/error code for when success=FALSE

`esp_ble_addr_type_t addr_type`
Peer device address type

`esp_bt_dev_type_t dev_type`
Device type

`esp_ble_auth_req_t auth_mode`
authentication mode

`struct esp_ble_gap_ext_adv_params_t`
ext adv parameters
Public Members

`esp_ble_ext_adv_type_mask_t type`
- ext adv type

`uint32_t interval_min`
- ext adv minimum interval

`uint32_t interval_max`
- ext adv maximum interval

`esp_ble_adv_channel_t channel_map`
- ext adv channel map

`esp_ble_addr_type_t own_addr_type`
- ext adv own address type

`esp_ble_addr_type_t peer_addr_type`
- ext adv peer address type

`esp_bd_addr_t peer_addr`
- ext adv peer address

`esp_ble_adv_filter_t filter_policy`
- ext adv filter policy

`int8_t tx_power`
- ext adv tx power

`esp_ble_gap_pri_phy_t primary_phy`
- ext adv primary phy

`uint8_t max_skip`
- ext adv maximum skip

`esp_ble_gap_phy_t secondary_phy`
- ext adv secondary phy

`uint8_t sid`
- ext adv sid

`bool scan_req_notif`
- ext adv scan request event notify

`struct esp_ble_ext_scan_cfg_t`
- ext scan config
Public Members

`esp_ble_scan_type_t scan_type`
- ext scan type

`uint16_t scan_interval`
- ext scan interval

`uint16_t scan_window`
- ext scan window

`struct esp_ble_ext_scan_params_t`
- ext scan parameters

Public Members

`esp_ble_addr_type_t own_addr_type`
- ext scan own address type

`esp_ble_scan_filter_t filter_policy`
- ext scan filter policy

`esp_ble_scan_duplicate_t scan_duplicate`
- ext scan duplicate scan

`esp_ble_ext_scan_cfg_mask_t cfg_mask`
- ext scan config mask

`esp_ble_ext_scan_cfg_t uncoded_cfg`
- ext scan uncoded config parameters

`esp_ble_ext_scan_cfg_t coded_cfg`
- ext scan coded config parameters

`struct esp_ble_gap_conn_params_t`
- create extend connection parameters

Public Members

`uint16_t scan_interval`
- init scan interval

`uint16_t scan_window`
- init scan window

`uint16_t interval_min`
- minimum interval
```c
uint16_t interval_max
maximum interval

uint16_t latency
ext scan type

uint16_t supervision_timeout
connection supervision timeout

uint16_t min_ce_len
minimum ce length

uint16_t max_ce_len
maximum ce length

struct esp_ble_gap_ext_adv_t
extend adv enable parameters

Public Members

uint8_t instance
advertising handle

int duration
advertising duration

int max_events
maximum number of extended advertising events

struct esp_ble_gap_periodic_adv_params_t
periodic adv parameters

Public Members

uint16_t interval_min
periodic advertising minimum interval

uint16_t interval_max
periodic advertising maximum interval

uint8_t properties
periodic advertising properties

struct esp_ble_gap_periodic_adv_sync_params_t
periodic adv sync parameters
```
Public Members

`esp_ble_gap_sync_t` *filter_policy*
periodic advertising sync filter policy

uint8_t *sid*
periodic advertising sid

`esp_ble_addr_type_t` *addr_type*
periodic advertising address type

`esp_bd_addr_t` *addr*
periodic advertising address

uint16_t *skip*
the maximum number of periodic advertising events that can be skipped

uint16_t *sync_timeout*
synchronization timeout

struct `esp_ble_gap_ext_adv_reprot_t`
extend adv report parameters

Public Members

`esp_ble_gap_adv_type_t` *event_type*
extend advertising type

uint8_t *addr_type*
extend advertising address type

`esp_bd_addr_t` *addr*
extend advertising address

`esp_ble_gap_pri_phy_t` *primary_phy*
extend advertising primary phy

`esp_ble_gap_phy_t` *secondly_phy*
extend advertising secondary phy

uint8_t *sid*
extend advertising sid

uint8_t *tx_power*
extend advertising tx power

int8_t *rssi*
extend advertising rssi
Chapter 2. API

uint16_t `per_adv_interval`
    periodic advertising interval

tuint8_t `dir_addr_type`
    direct address type

`esp_bd_addr_t dir_addr`
    direct address

`esp_ble_gap_ext_adv_data_status_t data_status`
    data type

uint8_t `adv_data_len`
    extend advertising data length

uint8_t `adv_data[251]`
    extend advertising data

struct `esp_ble_gap_periodic_adv_report_t`
    periodic adv report parameters

**Public Members**

uint16_t `sync_handle`
    periodic advertising train handle

uint8_t `tx_power`
    periodic advertising tx power

int8_t `rssi`
    periodic advertising rssi

`esp_ble_gap_ext_adv_data_status_t data_status`
    periodic advertising data type

uint8_t `data_length`
    periodic advertising data length

uint8_t `data[251]`
    periodic advertising data

struct `esp_ble_gap_periodic_adv_sync_estab_t`
    periodic adv sync establish parameters

**Public Members**
uint8_t status
    periodic advertising sync status

tuple16_t sync_handle
    periodic advertising train handle

uint8_t sid
    periodic advertising sid

esp_ble_addr_type_t addr_type
    periodic advertising address type

esp_bd_addr_t adv_addr
    periodic advertising address

esp_ble_gap_phy_t adv_phy
    periodic advertising adv phy type

uint16_t period_adv_interval
    periodic advertising interval

uint8_t adv_clk_accuracy
    periodic advertising clock accuracy

Macros

ESP_BLE_ADV_FLAG_LIMIT_DISC
    BLE_ADV_DATA_FLAG data flag bit definition used for advertising data flag.

ESP_BLE_ADV_FLAG_GEN_DISC

ESP_BLE_ADV_FLAG_BREDR_NOT_SPT

ESP_BLE_ADV_FLAG_DMT_CONTROLLER_SPT

ESP_BLE_ADV_FLAG_DMT_HOST_SPT

ESP_BLE_ADV_FLAG_NON_LIMIT_DISC

ESP_LE_KEY_NONE
    relate to BTM_LE_KEY_xxx in stack/btm_api.h
    No encryption key

ESP_LE_KEY_PENC
    encryption key, encryption information of peer device

ESP_LE_KEY_PID
    identity key of the peer device
**Chapter 2. API**

- **ESP_LE_KEY_PCSRK**
  peer SRK

- **ESP_LE_KEY_PLK**
  Link key

- **ESP_LE_KEY_LLK**
  peer link key

- **ESP_LE_KEY_LENC**
  master role security information

- **ESP_LE_KEY_LID**
  master device ID key

- **ESP_LE_KEY_LCSRK**
  local CSRK has been delivered to peer

- **ESP_LE_AUTH_NO_BOND**
  relate to BTM_LE_AUTH_xxx in stack/btm_api.h
  0 no bonding

- **ESP_LE_AUTH_BOND**
  1 « 0 device in the bonding with peer

- **ESP_LE_AUTH_REQ_MITM**
  1 « 2 man in the middle attack

- **ESP_LE_AUTH_REQ_BOND_MITM**
  0101 bonding with man in the middle attack

- **ESP_LE_AUTH_REQ_SC_ONLY**
  1 « 3 secure connection

- **ESP_LE_AUTH_REQ_SC_BOND**
  1001 secure connection with bond

- **ESP_LE_AUTH_REQ_SC_MITM**
  1100 secure conn with MITM

- **ESP_LE_AUTH_REQ_SC_MITM_BOND**
  1101 SC with MITM and Bonding

- **ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_DISABLE**
  authentication disable

- **ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_ENABLE**
  authentication enable
ESP_BLE_OOB_DISABLE
disbale the out of bond

ESP_BLE_OOB_ENABLE
enable the out of bond

ESP_IO_CAP_OUT
relate to BTM_IO_CAP_xxx in stack/btm_api.h
DisplayOnly

ESP_IO_CAP_IO
DisplayYesNo

ESP_IO_CAP_IN
KeyboardOnly

ESP_IO_CAP_NONE
NoInputNoOutput

ESP_IO_CAP_KBDISP
Keyboard display

ESP_BLE_APPEARANCE_UNKNOWN
relate to BTM_BLE_APPEARANCE_UNKNOWN in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_PHONE
relate to BTM_BLE_APPEARANCE_GENERIC_PHONE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_COMPUTER
relate to BTM_BLE_APPEARANCE_GENERIC_COMPUTER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_WATCH
relate to BTM_BLE_APPEARANCE_GENERIC_WATCH in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_SPORTS_WATCH
relate to BTM_BLE_APPEARANCE_SPORTS_WATCH in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_CLOCK
relate to BTM_BLE_APPEARANCE_GENERIC_CLOCK in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_DISPLAY
relate to BTM_BLE_APPEARANCE_GENERIC_DISPLAY in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_REMOTE
relate to BTM_BLE_APPEARANCE_GENERIC_REMOTE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_EYEGLASSES
relate to BTM_BLE_APPEARANCE_GENERIC_EYEGLASSES in stack/btm_ble_api.h
Chapter 2. API 参考

ESP_BLE_APPEARANCE_GENERIC_TAG
relate to BTM_BLE_APPEARANCE_GENERIC_TAG in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_KEYRING
relate to BTM_BLE_APPEARANCE_GENERIC_KEYRING in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_MEDIA_PLAYER
relate to BTM_BLE_APPEARANCE_GENERIC_MEDIA_PLAYER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_BARCODE_SCANNER
relate to BTM_BLE_APPEARANCE GENERIC BARCODE SCANNER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_THERMOMETER
relate to BTM_BLE_APPEARANCE GENERIC THERMOMETER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_THERMOMETER_EAR
relate to BTM_BLE_APPEARANCE THERMOMETER EAR in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_HEART_RATE
relate to BTM_BLE_APPEARANCE GENERIC HEART RATE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HEART_RATE_BELT
relate to BTM_BLE_APPEARANCE HEART RATE BELT in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_BLOOD_PRESSURE
relate to BTM_BLE_APPEARANCE GENERIC BLOOD PRESSURE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_BLOOD_PRESSURE_ARM
relate to BTM_BLE_APPEARANCE BLOOD PRESSURE ARM in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_BLOOD_PRESSURE_WRIST
relate to BTM_BLE_APPEARANCE BLOOD PRESSURE WRIST in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_HID
relate to BTM_BLE_APPEARANCE GENERIC HID in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_KEYBOARD
relate to BTM_BLE_APPEARANCE HID_KEYBOARD in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_MOUSE
relate to BTM_BLE_APPEARANCE HID_MOUSE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_JOYSTICK
relate to BTM_BLE_APPEARANCE HID_JOYSTICK in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_GAMEPAD
relate to BTM_BLE_APPEARANCE HID GAMEPAD in stack/btm_ble_api.h
ESP_BLE_APPEARANCE_HID_DIGITIZER_TABLET
  relate to BTM_BLE_APPEARANCE_HID_DIGITIZER_TABLET in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_CARD_READER
  relate to BTM_BLE_APPEARANCE_HID_CARD_READER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_DIGITAL_PEN
  relate to BTM_BLE_APPEARANCE_HID_DIGITAL_PEN in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_HID_BARCODE.Scanner
  relate to BTM_BLE_APPEARANCE_HID_BARCODE.Scanner in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_GLUCOSE
  relate to BTM_BLE_APPEARANCE_GENERIC_GLUCOSE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_WALKING
  relate to BTM_BLE_APPEARANCE_GENERIC_WALKING in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_WALKING_IN_SHOE
  relate to BTM_BLE_APPEARANCE_WALKING_IN_SHOE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_WALKING_ON_SHOE
  relate to BTM_BLE_APPEARANCE_WALKING_ON_SHOE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_WALKING_ON_HIP
  relate to BTM_BLE_APPEARANCE_WALKING_ON_HIP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_CYCLING
  relate to BTM_BLE_APPEARANCE_GENERIC_CYCLING in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_COMPUTER
  relate to BTM_BLE_APPEARANCE_CYCLING_COMPUTER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_SPEED
  relate to BTM_BLE_APPEARANCE_CYCLING_SPEED in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_CADENCE
  relate to BTM_BLE_APPEARANCE_CYCLING_CADENCE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_POWER
  relate to BTM_BLE_APPEARANCE_CYCLING_POWER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_CYCLING_SPEED_CADENCE
  relate to BTM_BLE_APPEARANCE_CYCLING_SPEED_CADENCE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_PULSE_OXIMETER
  relate to BTM_BLE_APPEARANCE_GENERIC_PULSE_OXIMETER in stack/btm_ble_api.h
ESP_BLE_APPEARANCE_PULSE_OXIMETER_FINGERTIP
relate to BTM_BLE_APPEARANCE_PULSE_OXIMETER_FINGERTIP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_PULSE_OXIMETER_WRIST
relate to BTM_BLE_APPEARANCE_PULSE_OXIMETER_WRIST in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_WEIGHT
relate to BTM_BLE_APPEARANCE_GENERIC_WEIGHT in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_PERSONAL_MOBILITY_DEVICE
relate to BTM_BLE_APPEARANCE_GENERIC_PERSONAL_MOBILITY_DEVICE in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_POWERED_WHEELCHAIR
relate to BTM_BLE_APPEARANCE_POWERED_WHEELCHAIR in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_MOBILITY_SCOOTER
relate to BTM_BLE_APPEARANCE_MOBILITY_SCOOTER in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MONITOR
relate to BTM_BLE_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MONITOR in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_INSULIN_PUMP
relate to BTM_BLE_APPEARANCE_GENERIC_INSULIN_PUMP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_INSULIN_PUMP_DURABLE_PUMP
relate to BTM_BLE_APPEARANCE_INSULIN_PUMP_DURABLE_PUMP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_INSULIN_PUMP_PATCH_PUMP
relate to BTM_BLE_APPEARANCE_INSULIN_PUMP_PATCH_PUMP in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_INSULIN_PEN
relate to BTM_BLE_APPEARANCE_INSULIN_PEN in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_MEDICATION_DELIVERY
relate to BTM_BLE_APPEARANCE_GENERIC_MEDICATION_DELIVERY in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_GENERIC_OUTDOOR_SPORTS
relate to BTM_BLE_APPEARANCE_GENERIC_OUTDOOR_SPORTS in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION
relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION in stack/btm_ble_api.h

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_AND_NAV
relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_AND_NAV in stack/btm_ble_api.h
**ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD**

Relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD in stack/btm_ble_api.h

**ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD_AND_NAV**

Relate to BTM_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD_AND_NAV in stack/btm_ble_api.h

**ESP_GAP_BLE_CHANNELS_LEN**

Channel length

**ESP_GAP_BLE_ADD_WHITELIST_COMPLETE_EVT**

This is the old name, just for backwards compatibility.

**ESP_BLE_ADV_DATA_LEN_MAX**

Advertising data maximum length.

**ESP_BLE_SCAN_RSP_DATA_LEN_MAX**

Scan response data maximum length.

**BLE_BIT (n)**

**ESP_BLE_GAP_SET_EXT_ADV_PROP_NONCONN_NONSCANNABLE_UNDIRECTED**

Non-Connectable and Non-Scannable Undirected advertising

**ESP_BLE_GAP_SET_EXT_ADV_PROP_CONNECTABLE**

Connectable advertising

**ESP_BLE_GAP_SET_EXT_ADV_PROP_SCANNABLE**

Scannable advertising

**ESP_BLE_GAP_SET_EXT_ADV_PROP_DIRECTED**

Directed advertising

**ESP_BLE_GAP_SET_EXT_ADV_PROP_HD_DIRECTED**

High Duty Cycle Directed Connectable advertising (<= 3.75 ms Advertising Interval)

**ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY**

Use legacy advertising PDUs

**ESP_BLE_GAP_SET_EXT_ADV_PROP_ANON_ADV**

Omit advertiser's address from all PDUs ("anonymous advertising")

**ESP_BLE_GAP_SET_EXT_ADV_PROP_INCLUDE_TX_PWR**

Include TxPower in the extended header of the advertising PDU

**ESP_BLE_GAP_SET_EXT_ADV_PROP_MASK**

Reserved for future use. If extended advertising PDU types are being used (bit 4 = 0) then: The advertisement shall not be both connectable and scannable. High duty cycle directed connectable advertising (<= 3.75 ms advertising interval) shall not be used (bit 3 = 0) ADV_IND
Chapter 2. API Reference

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_IND
    ADV_DIRECT_IND (low duty cycle)

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_LD_DIR
    ADV_DIRECT_IND (high duty cycle)

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_HD_DIR
    ADV_SCAN_IND

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_SCAN
    ADV_NONCONN_IND

ESP_BLE_GAP_SET_EXT_ADV_PROP_LEGACY_NONCONN

ESP_BLE_GAP_PHY_1M
    Secondary Advertisement PHY is LE1M

ESP_BLE_GAP_PHY_2M
    Secondary Advertisement PHY is LE2M

ESP_BLE_GAP_PHY_CODED
    Secondary Advertisement PHY is LE Coded

ESP_BLE_GAP_NO_PREFER_TRANSMIT_PHY
    No Prefer TX PHY supported by controller

ESP_BLE_GAP_NO_PREFER_RECEIVE_PHY
    No Prefer RX PHY supported by controller

ESP_BLE_GAP_PRI_PHY_1M
    Primary phy only support 1M and LE coded phy.
    Primary Phy is LE1M

ESP_BLE_GAP_PRI_PHY_CODED
    Primary Phy is LE CODED

ESP_BLE_GAP_PHY_1M_PREF_MASK
    The Host prefers use the LE1M transmitter or reciever PHY

ESP_BLE_GAP_PHY_2M_PREF_MASK
    The Host prefers use the LE2M transmitter or reciever PHY

ESP_BLE_GAP_PHY_CODED_PREF_MASK
    The Host prefers use the LE CODED transmitter or reciever PHY

ESP_BLE_GAP_PHY_OPTIONS_NO_PREF
    The Host has no preferred coding when transmitting on the LE Coded PHY
ESP_BLE_GAP_PHY_OPTIONS_PREF_S2_CODING
The Host prefers that S=2 coding be used when transmitting on the LE Coded PHY

ESP_BLE_GAP_PHY_OPTIONS_PREF_S8_CODING
The Host prefers that S=8 coding be used when transmitting on the LE Coded PHY

ESP_BLE_GAP_EXT_SCAN_CFG_UNCODE_MASK
Scan Advertisements on the LE1M PHY

ESP_BLE_GAP_EXT_SCAN_CFG_CODE_MASK
Scan advertisements on the LE coded PHY

ESP_BLE_GAP_EXT_ADV_DATA_COMPLETE
Advertising data.
extended advertising data compete

ESP_BLE_GAP_EXT_ADV_DATA_INCOMPLETE
extended advertising data incomplete

ESP_BLE_GAP_EXT_ADV_DATA_TRUNCATED
extended advertising data truncated mode

ESP_BLE_GAP_SYNC_POLICY_BY_ADV_INFO
Advertising SYNC policy.
sync policy by advertising info

ESP_BLE_GAP_SYNC_POLICY_BY.PERIODIC_LIST
periodic advertising sync policy

ESP_BLE_ADV_REPORT_EXT_ADV_IND
Advertising report.
advertising report with extended advertising indication type

ESP_BLE_ADV_REPORT_EXT.Scan_IND
advertising report with extended scan indication type

ESP_BLE_ADV_REPORT_EXT.DIRECT_ADV
advertising report with extended direct advertising indication type

ESP_BLE_ADV_REPORT_EXT.Scan_RSP
advertising report with extended scan response indication type Bluetooth 5.0, Vol 2, Part E, 7.7.65.13

ESP_BLE_LEGACY_ADV_TYPE_IND
advertising report with legacy advertising indication type

ESP_BLE_LEGACY_ADV_TYPE_DIRECT_IND
advertising report with legacy direct indication type
Chapter 2. API Reference

**ESP_BLE_LEGACY_ADV_TYPE_SCAN_IND**

advertising report with legacy scan indication type

**ESP_BLE_LEGACY_ADV_TYPE_NONCON_IND**

advertising report with legacy non-connectable indication type

**ESP_BLE_LEGACY_ADV_TYPE_SCAN_RSP_TO_ADV_IND**

advertising report with legacy scan response indication type

**ESP_BLE_LEGACY_ADV_TYPE_SCAN_RSP_TO_ADV_SCAN_IND**

advertising report with legacy advertising with scan response indication type

**EXT_ADV_TX_PWR_NO_PREFERENCE**

Extend advertising tx power, range: [-127, +126] dBm.

host has no preference for tx power

**Type Definitions**

typedef uint8_t esp_ble_key_type_t

typedef uint8_t esp_ble_auth_req_t

combination of the above bit pattern

typedef uint8_t esp_ble_io_cap_t

combination of the io capability

typedef uint8_t esp_gap_ble_channels[ESP_GAP_BLE_CHANNELS_LEN]

typedef uint8_t esp_duplicate_info_t[ESP_BD_ADDR_LEN]

typedef uint16_t esp_ble_ext_adv_type_mask_t

typedef uint8_t esp_ble_gap_phy_t

typedef uint8_t esp_ble_gap_all_phy_t

typedef uint8_t esp_ble_gap_pri_phy_t

typedef uint8_t esp_ble_gap_phy_mask_t

typedef uint16_t esp_ble_gap_prefer_phy_options_t

typedef uint8_t esp_ble_ext_scanCfg_mask_t

typedef uint8_t esp_ble_gap_ext_adv_data_status_t

typedef uint8_t esp_ble_gap_sync_t
typedef uint8_t esp_ble_gap_adv_type_t

typedef void (*esp_gap_ble_cb_t)(esp_gap_ble_cb_event_t event, esp_ble_gap_cb_param_t *param)

GAP callback function type.

- **Param event**: Event type
- **Param param**: Point to callback parameter, currently is union type

**Enumerations**

defined enum esp_gap_ble_cb_event_t

GAP BLE callback event type.

- **Values**:

  - enumerator ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT
    - When advertising data set complete, the event comes
  - enumerator ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT
    - When scan response data set complete, the event comes
  - enumerator ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT
    - When scan parameters set complete, the event comes
  - enumerator ESP_GAP_BLE_SCAN_RESULT_EVT
    - When one scan result ready, the event comes each time
  - enumerator ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT
    - When raw advertising data set complete, the event comes
  - enumerator ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT
    - When raw advertising data set complete, the event comes
  - enumerator ESP_GAP_BLE_ADV_START_COMPLETE_EVT
    - When start advertising complete, the event comes
  - enumerator ESP_GAP_BLE_SCAN_START_COMPLETE_EVT
    - When start scan complete, the event comes
  - enumerator ESP_GAP_BLE_AUTH_CMPL_EVT
    - Authentication complete indication.
  - enumerator ESP_GAP_BLE_KEY_EVT
    - BLE key event for peer device keys
  - enumerator ESP_GAP_BLE_SEC_REQ_EVT
    - BLE security request
  - enumerator ESP_GAP_BLE_PASSKEY_NOTIF_EVT
    - passkey notification event
enumerator **ESP_GAP_BLE_PASSKEY_REQ_EVT**
passkey request event

enumerator **ESP_GAP_BLE_OOB_REQ_EVT**
OOB request event

enumerator **ESP_GAP_BLE_LOCAL_IR_EVT**
BLE local IR (identity Root 128-bit random static value used to generate Long Term Key) event

enumerator **ESP_GAP_BLE_LOCAL_ER_EVT**
BLE local ER (Encryption Root value used to generate identity resolving key) event

enumerator **ESP_GAP_BLE_NC_REQ_EVT**
Numeric Comparison request event

enumerator **ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT**
When stop adv complete, the event comes

enumerator **ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT**
When stop scan complete, the event comes

enumerator **ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT**
When set the static rand address complete, the event comes

enumerator **ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT**
When update connection parameters complete, the event comes

enumerator **ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT**
When set pkt length complete, the event comes

enumerator **ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT**
When Enable/disable privacy on the local device complete, the event comes

enumerator **ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT**
When remove the bond device complete, the event comes

enumerator **ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT**
When clear the bond device clear complete, the event comes

enumerator **ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT**
When get the bond device list complete, the event comes

enumerator **ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT**
When read the rssi complete, the event comes

enumerator **ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT**
When add or remove whitelist complete, the event comes
enumerator ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT
   When update duplicate exceptional list complete, the event comes.

enumerator ESP_GAP_BLE_SET_CHANNELS_EVT
   When setting BLE channels complete, the event comes.

enumerator ESP_GAP_BLE_READ_PHY_COMPLETE_EVT
   When reading phy complete, this event comes.

enumerator ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT
   When preferred default phy complete, this event comes.

enumerator ESP_GAP_BLE_SET_PREFERRED_PHY_COMPLETE_EVT
   When preferred phy complete, this event comes.

enumerator ESP_GAP_BLE_EXT_ADV_SET_RAND_ADDR_COMPLETE_EVT
   When extended set random address complete, the event comes.

enumerator ESP_GAP_BLE_EXT_ADV_SET_PARAMS_COMPLETE_EVT
   When extended advertising parameter complete, the event comes.

enumerator ESP_GAP_BLE_EXT_ADV_DATA_SET_COMPLETE_EVT
   When extended advertising data complete, the event comes.

enumerator ESP_GAP_BLE_EXT_SCAN_RSP_DATA_SET_COMPLETE_EVT
   When extended scan response data complete, the event comes.

enumerator ESP_GAP_BLE_EXT_ADV_START_COMPLETE_EVT
   When extended advertising start complete, the event comes.

enumerator ESP_GAP_BLE_EXT_ADV_STOP_COMPLETE_EVT
   When extended advertising stop complete, the event comes.

enumerator ESP_GAP_BLE_EXT_ADV_SET_REMOVE_COMPLETE_EVT
   When extended advertising set remove complete, the event comes.

enumerator ESP_GAP_BLE_EXT_ADV_SET_CLEAR_COMPLETE_EVT
   When extended advertising set clear complete, the event comes.

enumerator ESP_GAP_BLE_PERIODIC_ADV_SET_PARAMS_COMPLETE_EVT
   When periodic advertising parameter complete, the event comes.

enumerator ESP_GAP_BLE_PERIODIC_ADV_DATA_SET_COMPLETE_EVT
   When periodic advertising data complete, the event comes.

enumerator ESP_GAP_BLE_PERIODIC_ADV_START_COMPLETE_EVT
   When periodic advertising start complete, the event comes.
enumerator ESP_GAP_BLE_PERIODIC_ADV_STOP_COMPLETE_EVT
    when periodic advertising stop complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_CREATE_SYNC_COMPLETE_EVT
    when periodic advertising create sync complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_SYNC_CANCEL_COMPLETE_EVT
    when extended advertising sync cancel complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_SYNC_TERMINATE_COMPLETE_EVT
    when extended advertising sync terminate complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_ADD_DEV_COMPLETE_EVT
    when extended advertising add device complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_REMOVE_DEV_COMPLETE_EVT
    when extended advertising remove device complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_CLEAR_DEV_COMPLETE_EVT
    when extended advertising clear device, the event comes

enumerator ESP_GAP_BLE_SET_EXT_SCAN_PARAMS_COMPLETE_EVT
    when extended scan parameter complete, the event comes

enumerator ESP_GAP_BLE_EXT_SCAN_START_COMPLETE_EVT
    when extended scan start complete, the event comes

enumerator ESP_GAP_BLE_EXT_SCAN_STOP_COMPLETE_EVT
    when extended scan stop complete, the event comes

enumerator ESP_GAP_BLE_PREFER_EXT_CONN_PARAMS_SET_COMPLETE_EVT
    when extended prefer connection parameter set complete, the event comes

enumerator ESP_GAP_BLE_PHY_UPDATE_COMPLETE_EVT
    when ble phy update complete, the event comes

enumerator ESP_GAP_BLE_EXT_ADV_REPORT_EVT
    when extended advertising report complete, the event comes

enumerator ESP_GAP_BLE_SCAN_TIMEOUT_EVT
    when scan timeout complete, the event comes

enumerator ESP_GAP_BLE_ADV_TERMINATED_EVT
    when advertising terminate data complete, the event comes

enumerator ESP_GAP_BLE_SCAN_REQ_RECEIVED_EVT
    when scan req received complete, the event comes
enumerator ESP_GAP_BLE_CHANNEL_SELECT_ALGORITHM_EVT
when channel select algorithm complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_REPORT_EVT
when periodic report advertising complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_SYNC_LOST_EVT
when periodic advertising sync lost complete, the event comes

enumerator ESP_GAP_BLE_PERIODIC_ADV_SYNC_ESTAB_EVT
when periodic advertising sync establish complete, the event comes

enumerator ESP_GAP_BLE_EVT_MAX
when maximum advertising event complete, the event comes

enum esp_ble_adv_data_type
The type of advertising data(not adv_type)

Values:

enumerator ESP_BLE_AD_TYPE_FLAG

enumerator ESP_BLE_AD_TYPE_16SRV_PART

enumerator ESP_BLE_AD_TYPE_16SRV_CMPL

enumerator ESP_BLE_AD_TYPE_32SRV_PART

enumerator ESP_BLE_AD_TYPE_32SRV_CMPL

enumerator ESP_BLE_AD_TYPE_128SRV_PART

enumerator ESP_BLE_AD_TYPE_128SRV_CMPL

enumerator ESP_BLE_AD_TYPE_NAME_SHORT

enumerator ESP_BLE_AD_TYPE_NAME_CMPL

enumerator ESP_BLE_AD_TYPE_TX_PWR

enumerator ESP_BLE_AD_TYPE_DEV_CLASS

enumerator ESP_BLE_AD_TYPE_SM_TK

enumerator ESP_BLE_AD_TYPE_SM_OOB_FLAG

enumerator ESP_BLE_AD_TYPE_INT_RANGE
enumerator ESP_BLE_AD_TYPE_SOL_SRV_UUID
enumerator ESP_BLE_AD_TYPE_128SOL_SRV_UUID
enumerator ESP_BLE_AD_TYPE_SERVICE_DATA
enumerator ESP_BLE_AD_TYPE_PUBLIC_TARGET
enumerator ESP_BLE_AD_TYPE_RANDOM_TARGET
enumerator ESP_BLE_AD_TYPE_APPEARANCE
enumerator ESP_BLE_AD_TYPE_ADV_INT
enumerator ESP_BLE_AD_TYPE_LE_DEV_ADDR
enumerator ESP_BLE_AD_TYPE_LE_ROLE
enumerator ESP_BLE_AD_TYPE_SPAIR_C256
enumerator ESP_BLE_AD_TYPE_SPAIR_R256
enumerator ESP_BLE_AD_TYPE_32SOL_SRV_UUID
enumerator ESP_BLE_AD_TYPE_32SERVICE_DATA
enumerator ESP_BLE_AD_TYPE_128SERVICE_DATA
enumerator ESP_BLE_AD_TYPE_LE_SECURE_CONFIRM
enumerator ESP_BLE_AD_TYPE_LE_SECURE_RANDOM
enumerator ESP_BLE_AD_TYPE_URI
enumerator ESP_BLE_AD_TYPE_INDOOR_POSITION
enumerator ESP_BLE_AD_TYPE_TRANS_DISC_DATA
enumerator ESP_BLE_AD_TYPE_LE_SUPPORT_FEATURE
enumerator ESP_BLE_AD_TYPE_CHAN_MAP_UPDATE
enumerator ESP_BLE_AD_TYPE_MANUFACTURER_SPECIFIC_TYPE
enum esp_ble_adv_type_t
   Advertising mode.
   Values:

   enumerator ADV_TYPE_IND

   enumerator ADV_TYPE_DIRECT_IND_HIGH

   enumerator ADV_TYPE_SCAN_IND

   enumerator ADV_TYPE_NONCONN_IND

   enumerator ADV_TYPE_DIRECT_IND_LOW

enum esp_ble_adv_channel_t
   Advertising channel mask.
   Values:

   enumerator ADV_CHNL_37

   enumerator ADV_CHNL_38

   enumerator ADV_CHNL_39

   enumerator ADV_CHNL_ALL

enum esp_ble_adv_filter_t
   Values:

   enumerator ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY
       Allow both scan and connection requests from anyone.

   enumerator ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
       Allow both scan req from White List devices only and connection req from anyone.

   enumerator ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
       Allow both scan req from anyone and connection req from White List devices only.

   enumerator ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST
       Allow scan and connection requests from White List devices only.

enum esp_ble_sec_act_t
   Values:

   enumerator ESP_BLE_SEC_ENCRYPT
Chapter 2. API 参考

relate to BTA_DM_BLE_SEC_ENCRYPT in bta/bta_api.h. If the device has already bonded, the stack will use Long Term Key (LTK) to encrypt with the remote device directly. Else if the device hasn’t bonded, the stack will use the default authentication request used the esp_ble_gap_set_security_param function set by the user.

enumerator ESP_BLE_SEC_ENCRYPT_NO_MITM
relate to BTA_DM_BLE_SEC_ENCRYPT_NO_MITM in bta/bta_api.h. If the device has been already bonded, the stack will check the LTK (Long Term Key) whether the authentication request has been met, and if met, use the LTK to encrypt with the remote device directly, else re-pair with the remote device. Else if the device hasn’t been bonded, the stack will use NO MITM authentication request in the current link instead of using the authreq in the esp_ble_gap_set_security_param function set by the user.

enumerator ESP_BLE_SEC_ENCRYPT_MITM
relate to BTA_DM_BLE_SEC_ENCRYPT_MITM in bta/bta_api.h. If the device has been already bonded, the stack will check the LTK (Long Term Key) whether the authentication request has been met, and if met, use the LTK to encrypt with the remote device directly, else re-pair with the remote device. Else if the device hasn’t been bonded, the stack will use MITM authentication request in the current link instead of using the authreq in the esp_ble_gap_set_security_param function set by the user.

enum esp_ble_sm_param_t

Values:

enumerator ESP_BLE_SM_PASSKEY
Authentication requirements of local device

enumerator ESP_BLE_SM_AUTHEN_REQ_MODE
The IO capability of local device

enumerator ESP_BLE_SM_IOCAP_MODE
Initiator Key Distribution/Generation

enumerator ESP_BLE_SM_SET_INIT_KEY
Responder Key Distribution/Generation

enumerator ESP_BLE_SM_SET_RSP_KEY
Maximum Encryption key size to support

enumerator ESP_BLE_SM_MAX_KEY_SIZE
Minimum Encryption key size requirement from Peer

enumerator ESP_BLE_SM_MIN_KEY_SIZE
Set static Passkey

enumerator ESP_BLE_SM_SET_STATIC_PASSKEY
Reset static Passkey

enumerator ESP_BLE_SM_CLEAR_STATIC_PASSKEY
Accept only specified SMP Authentication requirement
enumerator ESP_BLE_SM_ONLY_ACCEPT_SPECIFIED_SEC_AUTH
   Enable/Disable OOB support

enumerator ESP_BLE_SM_OOB_SUPPORT
   Appl encryption key size

enumerator ESP_BLE_APP_ENC_KEY_SIZE
   authentication max param

enumerator ESP_BLE_SM_MAX_PARAM

enum esp_ble_scan_type_t
   Ble scan type.
   Values:

enumerator BLE_SCAN_TYPE_PASSIVE
   Passive scan

enumerator BLE_SCAN_TYPE_ACTIVE
   Active scan

enum esp_ble_scan_filter_t
   Ble scan filter type.
   Values:

enumerator BLE_SCAN_FILTER_ALLOW_ALL
   Accept all:
      i. advertisement packets except directed advertising packets not addressed to this device (default).

enumerator BLE_SCAN_FILTER_ALLOW_ONLY_WLST
   Accept only:
      i. advertisement packets from devices where the advertiser’s address is in the White list.
      ii. Directed advertising packets which are not addressed for this device shall be ignored.

enumerator BLE_SCAN_FILTER_ALLOW_UND_RPA_DIR
   Accept all:
      i. undirected advertisement packets, and
      ii. directed advertising packets where the initiator address is a resolvable private address, and
      iii. directed advertising packets addressed to this device.

enumerator BLE_SCAN_FILTER_ALLOW_WLIST_RPA_DIR
   Accept all:
      i. advertisement packets from devices where the advertiser’s address is in the White list, and
      ii. directed advertising packets where the initiator address is a resolvable private address, and
      iii. directed advertising packets addressed to this device.
enum `esp_ble_scan_duplicate_t`

Ble scan duplicate type.

Values:

enumerator `BLE_SCAN_DUPLICATE_DISABLE`
the Link Layer should generate advertising reports to the host for each packet received

enumerator `BLE_SCAN_DUPLICATE_ENABLE`
the Link Layer should filter out duplicate advertising reports to the Host

enumerator `BLE_SCAN_DUPLICATE_MAX`
0x02 –0xFF, Reserved for future use

enum `esp_gap_search_evt_t`

Sub Event of ESP_GAP_BLE_SCAN_RESULT_EVT.

Values:

enumerator `ESP_GAP_SEARCH_INQ_RES_EVT`
Inquiry result for a peer device.

enumerator `ESP_GAP_SEARCH_INQ_CMPL_EVT`
Inquiry complete.

enumerator `ESP_GAP_SEARCH_DISC_RES_EVT`
Discovery result for a peer device.

enumerator `ESP_GAP_SEARCH_DISC_BLE_RES_EVT`
Discovery result for BLE GATT based service on a peer device.

enumerator `ESP_GAP_SEARCH_DISC_CMPL_EVT`
Discovery complete.

enumerator `ESP_GAP_SEARCH_DI_DISC_CMPL_EVT`
Discovery complete.

enumerator `ESP_GAP_SEARCH_SEARCH_CANCEL_CMPL_EVT`
Search cancelled

enumerator `ESP_GAP_SEARCH_INQ_DISCARD_NUM_EVT`
The number of pkt discarded by flow control

enum `esp_ble_evt_type_t`

Ble scan result event type, to indicate the result is scan response or advertising data or other.

Values:

enumerator `ESP_BLE_EVT_CONN_ADV`
Connectable undirected advertising (ADV_IND)
enumerator **ESP_BLE_EVT_CONN_DIR_ADV**
  Connectable directed advertising (ADV_DIRECT_IND)

enumerator **ESP_BLE_EVT_DISC_ADV**
  Scannable undirected advertising (ADV_SCAN_IND)

enumerator **ESP_BLE_EVT_NON_CONN_ADV**
  Non connectable undirected advertising (ADV_NONCONN_IND)

enumerator **ESP_BLE_EVT_SCAN_RSP**
  Scan Response (SCAN_RSP)

enum **esp_ble_wl_operation_t**

  Values:

  enumerator **ESP_BLE_WHITELIST_REMOVE**
    remove mac from whitelist

 enumerator **ESP_BLE_WHITELIST_ADD**
    add address to whitelist

enum **esp_bt_duplicate_exceptional_subcode_type_t**

  Values:

  enumerator **ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_ADD**
    Add device info into duplicate scan exceptional list

  enumerator **ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_REMOVE**
    Remove device info from duplicate scan exceptional list

  enumerator **ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_CLEAN**
    Clean duplicate scan exceptional list

enum **esp_ble_duplicate_exceptional_info_type_t**

  Values:

  enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_ADV_ADDR**
    BLE advertising address, device info will be added into ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST

  enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_LINK_ID**
    BLE mesh link ID, it is for BLE mesh, device info will be added into ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST

  enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_BEACON_TYPE**
    BLE mesh beacon AD type, the format is |Len|0x2B|Beacon Type|Beacon Data|

  enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROV_SRV_ADV**
    BLE mesh provisioning service uuid, the format is |0x02|0x01|flags|0x03|0x03|0x1827|... |
enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROXY_SRV_ADV**

BLE mesh adv with proxy service uuid, the format is |0x02|0x01|flags|0x03|0x03|0x1828|… |`

enum esp_duplicate_scan_exceptional_list_type_t

Values:

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST**

duplicate scan exceptional addr list

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST**

duplicate scan exceptional mesh link ID list

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_BEACON_TYPE_LIST**

duplicate scan exceptional mesh beacon type list

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROV_SRV_ADV_LIST**

duplicate scan exceptional mesh adv with provisioning service uuid

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROXY_SRV_ADV_LIST**

duplicate scan exceptional mesh adv with provisioning service uuid

enumerator **ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ALL_LIST**

duplicate scan exceptional all list

GATT DEFINES

API Reference

Header File

- components/bluetooth/bluedroid/api/include/api/esp_gatt_defs.h

Unions

union esp_gatt_rsp_t

```
#include <esp_gatt_defs.h> GATT remote read request response type.
```

Public Members

```
esp_gatt_value_t attr_value

Gatt attribute structure
```

```
uint16_t handle

Gatt attribute handle
```

Structures

struct esp_gatt_id_t

Gatt id, include uuid and instance id.
Chapter 2. API

**Public Members**

```c

esp_bt_uuid_t uuid
    UUID

uint8_t inst_id
    Instance id

struct esp_gatt_srvc_id_t
    Gatt service id, include id (uuid and instance id) and primary flag.

esp_gatt_id_t id
    Gatt id, include uuid and instance

bool is_primary
    This service is primary or not

struct esp_attr_desc_t
    Attribute description (used to create database)

uint16_t uuid_length
    UUID length

uint8_t *uuid_p
    UUID value

uint16_t perm
    Attribute permission

uint16_t max_length
    Maximum length of the element

uint16_t length
    Current length of the element

uint8_t *value
    Element value array

struct esp_attr_control_t
    attribute auto response flag
```
chapter 2. API

public members

uint8_t auto_rsp

if auto_rsp set to ESP_GATT_RSP_BY_APP, means the response of Write/Read operation will be replied by application. if auto_rsp set to ESP_GATT_AUTO_RSP, means the response of Write/Read operation will be replied by GATT stack automatically.

struct esp_gatts_attr_db_t

attribute type added to the gatt server database

public members

esp_attr_control_t attr_control

the attribute control type

esp_attr_desc_t att_desc

the attribute type

struct esp_attr_value_t

set the attribute value type

public members

uint16_t attr_max_len

attribute max value length

uint16_t attr_len

attribute current value length

uint8_t *attr_value

the pointer to attribute value

struct esp_gatts_incl_svc_desc_t

Gatt include service entry element.

public members

uint16_t start_hdl

Gatt start handle value of included service

uint16_t end_hdl

Gatt end handle value of included service

uint16_t uuid

Gatt attribute value UUID of included service

struct esp_gatts_incl128_svc_desc_t

Gatt include 128 bit service entry element.
**Public Members**

`uint16_t start_hdl`
- Gatt start handle value of included 128 bit service

`uint16_t end_hdl`
- Gatt end handle value of included 128 bit service

```c
struct esp_gatt_value_t
    Gatt attribute value.
```

**Public Members**

`uint8_t value[ESP_GATT_MAX_ATTR_LEN]`
- Gatt attribute value

`uint16_t handle`
- Gatt attribute handle

`uint16_t offset`
- Gatt attribute value offset

`uint16_t len`
- Gatt attribute value length

`uint8_t auth_req`
- Gatt authentication request

```c
struct esp_gatt_conn_params_t
    Connection parameters information.
```

**Public Members**

`uint16_t interval`
- connection interval

`uint16_t latency`
- Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

`uint16_t timeout`
- Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80
  Time = N * 10 msec Time Range: 100 msec to 32 seconds

```c
struct esp_gattc_multi_t
    read multiple attribute
```
Chapter 2. API

Public Members

uint8_t num_attr
  The number of the attribute

uint16_t handles[ESP_GATT_MAX_READ_MULTI_HANDLES]
  The handles list

struct esp_gattc_db_elem_t
data base attribute element

Public Members

esp_gatt_db_attr_type_t type
  The attribute type

uint16_t attribute_handle
  The attribute handle, it’s valid for all of the type

uint16_t start_handle
  The service start handle, it’s valid only when the type = ESP_GATT_DB_PRIMARY_SERVICE or ESP_GATT_DB_SECONDARY_SERVICE

uint16_t end_handle
  The service end handle, it’s valid only when the type = ESP_GATT_DB_PRIMARY_SERVICE or ESP_GATT_DB_SECONDARY_SERVICE

esp_gatt_char_prop_t properties
  The characteristic properties, it’s valid only when the type = ESP_GATT_DB_CHARACTERISTIC

esp.bt_uuid_t uuid
  The attribute uuid, it’s valid for all of the type

struct esp_gattc_service_elem_t
  service element

Public Members

bool is_primary
  The service flag, true if the service is primary service, else is secondary service

uint16_t start_handle
  The start handle of the service

uint16_t end_handle
  The end handle of the service
The uuid of the service

**struct esp_gattc_char_elem_t**
characteristic element

**Public Members**

```c
uint16_t char_handle
The characteristic handle
```

```c
esp_gatt_char_prop_t properties
The characteristic properties
```

```c
esp_bt_uuid_t uuid
The characteristic uuid
```

**struct esp_gattc_descr_elem_t**
descriptor element

**Public Members**

```c
uint16_t handle
The characteristic descriptor handle
```

```c
esp_bt_uuid_t uuid
The characteristic descriptor uuid
```

**struct esp_gattc_incl_svc_elem_t**
include service element

**Public Members**

```c
uint16_t handle
The include service current attribute handle
```

```c
uint16_t incl_srvc_s_handle
The start handle of the service which has been included
```

```c
uint16_t incl_srvc_e_handle
The end handle of the service which has been included
```

```c
esp_bt_uuid_t uuid
The include service uuid
```
Chapter 2. API Reference

**Macros**

- ESP_GATT_UUID_IMMEDIATE_ALERT_SVC
- ESP_GATT_UUID_LINK_LOSS_SVC
- ESP_GATT_UUID_TX_POWER_SVC
- ESP_GATT_UUID_CURRENT_TIME_SVC
- ESP_GATT_UUID_REF_TIME_UPDATE_SVC
- ESP_GATT_UUID_NEXT_DST_CHANGE_SVC
- ESP_GATT_UUID_GLUCOSE_SVC
- ESP_GATT_UUID_HEALTH_THERMOM_SVC
- ESP_GATT_UUID_DEVICE_INFO_SVC
- ESP_GATT_UUID_HEART_RATE_SVC
- ESP_GATT_UUID_PHONE_ALERT_STATUS_SVC
- ESP_GATT_UUID_BATTERY_SERVICE_SVC
- ESP_GATT_UUID_BLOOD_PRESSURE_SVC
- ESP_GATT_UUID_ALERT_NTF_SVC
- ESP_GATT_UUID_HID_SVC
- ESP_GATT_UUID_SCAN_PARAMETERS_SVC
- ESP_GATT_UUID_RUNNING_SPEED_CADENCE_SVC
- ESP_GATT_UUID_Automation_IO_SVC
- ESP_GATT_UUID_CYCLING_SPEED_CADENCE_SVC
- ESP_GATT_UUID_CYCLING_POWER_SVC
- ESP_GATT_UUID_LOCATION_AND_NAVIGATION_SVC
- ESP_GATT_UUID_ENVIRONMENTAL_SENSING_SVC

All "ESP_GATT_UUID_xxx" is attribute types
ESP_GATT_UUID_BODY_COMPOSITION

ESP_GATT_UUID_USER_DATA_SVC

ESP_GATT_UUID_WEIGHT_SCALE_SVC

ESP_GATT_UUID_BOND_MANAGEMENT_SVC

ESP_GATT_UUID_CONT_GLUCOSE_MONITOR_SVC

ESP_GATT_UUID_PRI_SERVICE

ESP_GATT_UUID_SEC_SERVICE

ESP_GATT_UUID_INCLUDE_SERVICE

ESP_GATT_UUID_CHAR_DECLARE

ESP_GATT_UUID_CHAR_EXT_PROP

ESP_GATT_UUID_CHAR_DESCRIPTION

ESP_GATT_UUID_CHAR_CLIENT_CONFIG

ESP_GATT_UUID_CHAR_SRVR_CONFIG

ESP_GATT_UUID_CHAR_PRESENT_FORMAT

ESP_GATT_UUID_CHAR_AGG_FORMAT

ESP_GATT_UUID_CHAR_VALID_RANGE

ESP_GATT_UUID_EXT_RPT_REF_DESCR

ESP_GATT_UUID_RPT_REF_DESCR

ESP_GATT_UUID_NUM_DIGITALS_DESCR

ESP_GATT_UUID_VALUE_TRIGGER_DESCR

ESP_GATT_UUID_ENV_SENSING_CONFIG_DESCR

ESP_GATT_UUID_ENV_SENSING_MEASUREMENT_DESCR

ESP_GATT_UUID_ENV_SENSING_TRIGGER_DESCR
ESP_GATT_UUID_TIME_TRIGGER_DESCR
ESP_GATT_UUID_GAP_DEVICE_NAME
ESP_GATT_UUID_GAP_ICON
ESP_GATT_UUID_GAP_PREF_CONN_PARAM
ESP_GATT_UUID_GAP_CENTRAL_ADDR_RESOL
ESP_GATT_UUID_GATT_SRV_CHGD
ESP_GATT_UUID_ALERT_LEVEL
ESP_GATT_UUID_TX_POWER_LEVEL
ESP_GATT_UUID_CURRENT_TIME
ESP_GATT_UUID_LOCAL_TIME_INFO
ESP_GATT_UUID_REF_TIME_INFO
ESP_GATT_UUID_NW_STATUS
ESP_GATT_UUID_NW_TRIGGER
ESP_GATT_UUID_ALERT_STATUS
ESP_GATT_UUID_RINGER_CP
ESP_GATT_UUID_RINGER_SETTING
ESP_GATT_UUID_GM_MEASUREMENT
ESP_GATT_UUID_GM_CONTEXT
ESP_GATT_UUID_GM_CONTROL_POINT
ESP_GATT_UUID_GM_FEATURE
ESP_GATT_UUID_SYSTEM_ID
ESP_GATT_UUID_MODEL_NUMBER_STR
ESP_GATT_UUID_SERIAL_NUMBER_STR
ESP_GATT_UUID_FW_VERSION_STR
ESP_GATT_UUID_HW_VERSION_STR
ESP_GATT_UUID_SW_VERSION_STR
ESP_GATT_UUID_MANU_NAME
ESP_GATT_UUID_IEEE_DATA
ESP_GATT_UUID_PNP_ID
ESP_GATT_UUID_HID_INFORMATION
ESP_GATT_UUID_HID_REPORT_MAP
ESP_GATT_UUID_HID_CONTROL_POINT
ESP_GATT_UUID_HID_REPORT
ESP_GATT_UUID_HID_PROTO_MODE
ESP_GATT_UUID_HID_BT_KB_INPUT
ESP_GATT_UUID_HID_BT_KB_OUTPUT
ESP_GATT_UUID_HID_BT_MOUSE_INPUT
ESP_GATT_HEART_RATE_MEAS
  Heart Rate Measurement.
ESP_GATT_BODY_SENSOR_LOCATION
  Body Sensor Location.
ESP_GATT_HEART_RATE_CNTL_POINT
  Heart Rate Control Point.
ESP_GATT_UUID_BATTERY_LEVEL
ESP_GATT_UUID_SC_CONTROL_POINT
ESP_GATT_UUID_SENSOR_LOCATION
ESP_GATT_UUID_RSC_MEASUREMENT
ESP_GATT_UUID_RSC_FEATURE
Chapter 2. API 参考

ESP_GATT_UUID_CSC_MEASUREMENT
ESP_GATT_UUID_CSC_FEATURE
ESP_GATT_UUID_SCAN_INT_WINDOW
ESP_GATT_UUID_SCAN_REFRESH
ESP_GATT_ILLEGAL_UUID
    GATT INVALID UUID.
ESP_GATT_ILLEGAL_HANDLE
    GATT INVALID HANDLE.
ESP_GATT_ATTR_HANDLE_MAX
    GATT attribute max handle.
ESP_GATT_MAX_READ_MULTI_HANDLES
ESP_GATT_PERM_READ
    Attribute permissions.
ESP_GATT_PERM_READ_ENCRYPTED
ESP_GATT_PERM_READ_ENC_MITM
ESP_GATT_PERM_WRITE
ESP_GATT_PERM_WRITE_ENCRYPTED
ESP_GATT_PERM_WRITE_ENC_MITM
ESP_GATT_PERM_WRITE_SIGNED
ESP_GATT_PERM_WRITE_SIGNED_MITM
ESP_GATT_PERM_READ_AUTHORIZATION
ESP_GATT_PERM_WRITE_AUTHORIZATION
ESP_GATT_CHAR_PROP_BIT_BROADCAST
ESP_GATT_CHAR_PROP_BIT_READ
ESP_GATT_CHAR_PROP_BIT_WRITE_NR
ESP_GATT_CHAR_PROP_BIT_WRITE
ESP_GATT_CHAR_PROP_BIT_NOTIFY
ESP_GATT_CHAR_PROP_BIT_INDIQUE
ESP_GATT_CHAR_PROP_BIT_AUTH
ESP_GATT_CHAR_PROP_BIT_EXT_PROP

ESP_GATT_MAX_ATTR_LEN
GATT maximum attribute length.

ESP_GATT_RSP_BY_APP
ESP_GATT_AUTO_RSP

ESP_GATT_IF_NONE
If callback report gattc_if/gatts_if as this macro, means this event is not correspond to any app

Type Definitions
typedef uint16_t esp_gatt_perm_t
typedef uint8_t esp_gatt_char_prop_t
typedef uint8_t esp_gatt_if_t

Gatt interface type, different application on GATT client use different gatt_if

Enumerations
enum esp_gatt_prep_write_type
Attribute write data type from the client.
Values:

enumerator ESP_GATT_PREP_WRITE_CANCEL
Prepare write cancel

enumerator ESP_GATT_PREP_WRITE_EXEC
Prepare write execute

enum esp_gatt_status_t
GATT success code and error codes.
Values:

enumerator ESP_GATT_OK
enumerator ESP_GATT_INVALID_HANDLE
enumerator ESP_GATT_READ_NOT_PERMIT
enumerator ESP_GATT_WRITE_NOT_PERMIT
enumerator ESP_GATT_INVALID_PDU
enumerator ESP_GATT_INSUF_AUTHENTICATION
enumerator ESP_GATT_REQ_NOT_SUPPORTED
enumerator ESP_GATT_INVALID_OFFSET
enumerator ESP_GATT_INSUFAuthorization
enumerator ESP_GATT_PREPARE_Q_FULL
enumerator ESP_GATT_NOT_FOUND
enumerator ESP_GATT_NOT_LONG
enumerator ESP_GATT_INSUF_KEY_SIZE
enumerator ESP_GATT_INVALID_ATTR_LEN
enumerator ESP_GATT_ERR_UNLIKELY
enumerator ESP_GATT_INSUF_ENCRYPTION
enumerator ESP_GATT_UNSUPPORT_GRP_TYPE
enumerator ESP_GATT_INSUF_RESOURCE
enumerator ESP_GATT_NO_RESOURCES
enumerator ESP_GATT_INTERNAL_ERROR
enumerator ESP_GATT_WRONG_STATE
enumerator ESP_GATT_DB_FULL
enumerator ESP_GATT_BUSY
enumerator ESP_GATT_ERROR
enumerator ESP_GATT_CMD_STARTED
enumerator ESP_GATT_ILLEGAL_PARAMETER
enumerator ESP_GATT_PENDING
enumerator ESP_GATT_AUTH_FAIL
enumerator ESP_GATT_MORE
enumerator ESP_GATT_INVALID_CFG
enumerator ESP_GATT_SERVICE_STARTED
enumerator ESP_GATT_ENCRYPTED_MITM
enumerator ESP_GATT_ENCRYPTED_NO_MITM
enumerator ESP_GATT_NOT_ENCRYPTED
enumerator ESP_GATT_CONGESTED
enumerator ESP_GATT_DUP_REG
enumerator ESP_GATT_ALREADY_OPEN
enumerator ESP_GATT_CANCEL
enumerator ESP_GATT_STACK_RSP
enumerator ESP_GATT_APP_RSP
enumerator ESP_GATT_UNKNOWN_ERROR
enumerator ESP_GATT_CCC_CFG_ERR
enumerator ESP_GATT_PRC_IN_PROGRESS
enumerator ESP_GATT_OUT_OF_RANGE

enum esp_gatt_conn_reason_t
    Gatt Connection reason enum.
    Values:
        enumerator ESP_GATT_CONN_UNKNOWN
            Gatt connection unknown
enumerator ESP_GATT_CONN_L2C_FAILURE
       General L2cap failure

enumerator ESP_GATT_CONN_TIMEOUT
       Connection timeout

enumerator ESP_GATT_CONN_TERMINATE_PEER_USER
       Connection terminate by peer user

enumerator ESP_GATT_CONN_TERMINATE_LOCAL_HOST
       Connection terminated by local host

enumerator ESP_GATT_CONN_FAIL_ESTABLISH
       Connection fail to establish

enumerator ESP_GATT_CONN_LMP_TIMEOUT
       Connection fail for LMP response tout

enumerator ESP_GATT_CONN_CONN_CANCEL
       L2CAP connection cancelled

enumerator ESP_GATT_CONN_NONE
       No connection to cancel

enum esp_gatt_auth_req_t
       Gatt authentication request type.

Values:

enumerator ESP_GATT_AUTH_REQ_NONE

enumerator ESP_GATT_AUTH_REQ_NO_MITM

enumerator ESP_GATT_AUTH_REQ_MITM

enumerator ESP_GATT_AUTH_REQ_SIGNED_NO_MITM

enumerator ESP_GATT_AUTH_REQ_SIGNED_MITM

enum esp_service_source_t

Values:

enumerator ESP_GATT_SERVICE_FROM_REMOTE_DEVICE

enumerator ESP_GATT_SERVICE_FROM_NVS_FLASH

enumerator ESP_GATT_SERVICE_FROM_UNKNOWN
enum **esp_gatt_write_type_t**
Gatt write type.

*Values:*

enumerator **ESP_GATT_WRITE_TYPE_NO_RSP**
Gatt write attribute need no response

enumerator **ESP_GATT_WRITE_TYPE_RSP**
Gatt write attribute need remote response

enum **esp_gatt_db_attr_type_t**
the type of attribute element

*Values:*

enumerator **ESP_GATT_DB_PRIMARY_SERVICE**
Gattc primary service attribute type in the cache

enumerator **ESP_GATT_DB_SECONDARY_SERVICE**
Gattc secondary service attribute type in the cache

enumerator **ESP_GATT_DB_CHARACTERISTIC**
Gattc characteristic attribute type in the cache

enumerator **ESP_GATT_DB_DESCRIPTOR**
Gattc characteristic descriptor attribute type in the cache

enumerator **ESP_GATT_DB_INCLUDED_SERVICE**
Gattc include service attribute type in the cache

enumerator **ESP_GATT_DB_ALL**
Gattc all the attribute (primary service & secondary service & include service & char & descriptor) type in the cache

**GATT SERVER API**

**Application Example**
Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a GATT sever demo and its tutorial. This demo creates a GATT service with an attribute table, which releases the user from adding attributes one by one. This is the recommended method of adding attributes.
  - bluetooth/bluedroid/ble/gatt_server_service_table
  - GATT Server Service Table Example Walkthrough
- This is a GATT server demo and its tutorial. This demo creates a GATT service by adding attributes one by one as defined by Bluedroid. The recommended method of adding attributes is presented in example above.
  - bluetooth/bluedroid/ble/gatt_server
  - GATT Server Example Walkthrough
- This is a BLE SPP-Like demo. This demo, which acts as a GATT server, can receive data from UART and then send the data to the peer device automatically.
  - bluetooth/bluedroid/ble/ble_spp_server

**API Reference**
Header File

- components/bt/host/bluedroid/api/include/api/esp_gatts_api.h

Functions

`esp_err_t esp_ble_gatts_register_callback(esp_gatts_cb_t callback)`

This function is called to register application callbacks with BTA GATTS module.

- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gatts_app_register(uint16_t app_id)`

This function is called to register application identifier.

- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gatts_app_unregister(esp_gatt_if_t gatts_if)`

unregister with GATT Server.

- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gatts_create_service(esp_gatt_if_t gatts_if, esp_gatt_srvc_id_t *service_id, uint16_t num_handle)`

Create a service. When service creation is done, a callback event ESP_GATTS_CREATE_EVT is called to report status and service ID to the profile. The service ID obtained in the callback function needs to be used when adding included service and characteristics/descriptors into the service.

- gatts_if [in] GATT server access interface
- service_id [in] service ID.
- num_handle [in] number of handle requested for this service.

- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gatts_create_attr_tab(const esp_gatts_attr_db_t *gatts_attr_db, esp_gatt_if_t gatts_if, uint16_t max_nb_attr, uint8_t srvc_inst_id)`

Create a service attribute tab.

- gatts_attr_db [in] the pointer to the service attr tab
- gatts_if [in] GATT server access interface
- max_nb_attr [in] the number of attribute to be added to the service database.
- srvc_inst_id [in] the instance id of the service

- ESP_OK: success
- other: failed

`esp_err_t esp_ble_gatts_add_included_service(uint16_t service_handle, uint16_t included_service_handle)`

This function is called to add an included service. This function have to be called between ‘esp_ble_gatts_create_service’ and ‘esp_ble_gatts_add_char’. After included service is included, a callback event ESP_GATTS_ADD_INCL_SVRC_EVT is reported the included service ID.

- service_handle [in] service handle to which this included service is to be added.
include_service_handle [in] the service ID to be included.

- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gatts_add_char(uint16_t service_handle, esp_bt_uuid_t *char_uuid, esp_gatt_perm_t perm, esp_gatt_char_prop_t property, esp_attr_value_t *char_val, esp_attr_control_t *control)
```

This function is called to add a characteristic into a service.

**Parameters**

- `service_handle` [in] - service handle to which this included service is to be added.
- `char_uuid` [in] - Characteristic UUID.
- `perm` [in] - Characteristic value declaration attribute permission.
- `property` [in] - Characteristic Properties
- `char_val` [in] - Characteristic value
- `control` [in] - attribute response control byte

**Return**

- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gatts_add_char_descr(uint16_t service_handle, esp_bt_uuid_t *descr_uuid, esp_gatt_perm_t perm, esp_attr_value_t *char_descr_val, esp_attr_control_t *control)
```

This function is called to add characteristic descriptor. When it’s done, a callback event ESP_GATT.S_ADD_DESCR_EVT is called to report the status and an ID number for this descriptor.

**Parameters**

- `service_handle` [in] - service handle to which this characteristic descriptor is to be added.
- `perm` [in] - descriptor access permission.
- `descr_uuid` [in] - descriptor UUID.
- `char_descr_val` [in] - Characteristic descriptor value
- `control` [in] - attribute response control byte

**Return**

- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gatts_delete_service(uint16_t service_handle)
```

This function is called to delete a service. When this is done, a callback event ESP_GATT.S_DELETE_EVT is report with the status.

**Parameters**

- `service_handle` [in] - service handle to be deleted.

**Return**

- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gatts_start_service(uint16_t service_handle)
```

This function is called to start a service.

**Parameters**

- `service_handle` [in] - the service handle to be started.

**Return**

- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gatts_stop_service(uint16_t service_handle)
```

This function is called to stop a service.

**Parameters**

- `service_handle` [in] - service to be toppled.

**Return**

- ESP_OK: success
- other: failed
esp_err_t esp_ble_gatts_send_indicate (esp_gatt_if_t gatts_if, uint16_t conn_id, uint16_t attr_handle, uint16_t value_len, uint8_t* value, bool need_confirm)

Send indicate or notify to GATT client. Set param need_confirm as false will send notification, otherwise indication.

参数
- gatts_if - [in] GATT server access interface
- conn_id - [in] - connection id to indicate.
- attr_handle - [in] - attribute handle to indicate.
- value_len - [in] - indicate value length.
- value - [in] - value to indicate.
- need_confirm - [in] - Whether a confirmation is required. false sends a GATT notification, true sends a GATT indication.

返回
- ESP_OK : success
- other : failed

esp_err_t esp_ble_gatts_send_response (esp_gatt_if_t gatts_if, uint16_t conn_id, uint32_t trans_id, esp_gatt_status_t status, esp_gatt_rsp_t *rsp)

This function is called to send a response to a request.

参数
- gatts_if - [in] GATT server access interface
- trans_id - [in] - transfer id
- status - [in] - response status
- rsp - [in] - response data.

返回
- ESP_OK : success
- other : failed

esp_err_t esp_ble_gatts_set_attr_value (uint16_t attr_handle, uint16_t length, const uint8_t* value)

This function is called to set the attribute value by the application.

参数
- attr_handle - [in] the attribute handle which to be set
- length - [in] the value length
- value - [in] the pointer to the attribute value

返回
- ESP_OK : success
- other : failed

esp_gatt_status_t esp_ble_gatts_get_attr_value (uint16_t attr_handle, uint16_t *length, const uint8_t **value)

Retrieve attribute value.

参数
- attr_handle - [in] Attribute handle.
- length - [out] pointer to the attribute value length
- value - [out] Pointer to attribute value payload, the value cannot be modified by user

返回
- ESP_GATT_OK : success
- other : failed

esp_err_t esp_ble_gatts_open (esp_gatt_if_t gatts_if, esp_bd_addr_t remote_bda, bool is_direct)

Open a direct open connection or add a background auto connection.

参数
- gatts_if - [in] GATT server access interface
- remote_bda - [in] remote device bluetooth device address.
- is_direct - [in] direct connection or background auto connection
Chapter 2. API 参考

返回
• ESP_OK : success
• other : failed

```c
esp_err_t esp_ble_gatts_close(esp_gatt_if_t gatts_if, uint16_t conn_id)
```

Close a connection a remote device.

参数
• gatts_if - [in] GATT server access interface
• conn_id - [in] connection ID to be closed.

返回
• ESP_OK : success
• other : failed

```c
esp_err_t esp_ble_gatts_send_service_change_indication(esp_gatt_if_t gatts_if, esp_bd_addr_t remote_bda)
```

Send service change indication.

参数
• gatts_if - [in] GATT server access interface
• remote_bda - [in] remote device bluetooth device address. If remote_bda is NULL then it will send service change indication to all the connected devices and if not then to a specific device

返回
• ESP_OK : success
• other : failed

Unions

union esp_ble_gatts_cb_param_t

```c
#include <esp_gatts_api.h> Gatt server callback parameters union.
```

Public Members

struct esp_ble_gatts_cb_param_t::gatts_reg_evt_param reg

Gatt server callback param of ESP_GATTS_REG_EVT

struct esp_ble_gatts_cb_param_t::gatts_read_evt_param read

Gatt server callback param of ESP_GATTS_READ_EVT

struct esp_ble_gatts_cb_param_t::gatts_write_evt_param write

Gatt server callback param of ESP_GATTS_WRITE_EVT

struct esp_ble_gatts_cb_param_t::gatts_exec_write_evt_param exec_write

Gatt server callback param of ESP_GATTS_EXEC_WRITE_EVT

struct esp_ble_gatts_cb_param_t::gatts_mtu_evt_param mtu

Gatt server callback param of ESP_GATTS_MTU_EVT

struct esp_ble_gatts_cb_param_t::gatts_conf_evt_param conf

Gatt server callback param of ESP_GATTS_CONF_EVT (confirm)

struct esp_ble_gatts_cb_param_t::gatts_create_evt_param create

Gatt server callback param of ESP_GATTS_CREATE_EVT
struct esp_ble_gatts_cb_param_t::gatts_add_incl_srvc_evt_param add_incl_srvc
    Gatt server callback param of ESP_GATTS_ADD_INCL_SRVC_EVT

struct esp_ble_gatts_cb_param_t::gatts_add_char_evt_param add_char
    Gatt server callback param of ESP_GATTS_ADD_CHAR_EVT

struct esp_ble_gatts_cb_param_t::gatts_add_char_descr_evt_param add_char_descr
    Gatt server callback param of ESP_GATTS_ADD_CHAR_DESCR_EVT

struct esp_ble_gatts_cb_param_t::gatts_delete_evt_param del
    Gatt server callback param of ESP_GATTS_DELETE_EVT

struct esp_ble_gatts_cb_param_t::gatts_start_evt_param start
    Gatt server callback param of ESP_GATTS_START_EVT

struct esp_ble_gatts_cb_param_t::gatts_stop_evt_param stop
    Gatt server callback param of ESP_GATTS_STOP_EVT

struct esp_ble_gatts_cb_param_t::gatts_connect_evt_param connect
    Gatt server callback param of ESP_GATTS_CONNECT_EVT

struct esp_ble_gatts_cb_param_t::gatts_disconnect_evt_param disconnect
    Gatt server callback param of ESP_GATTS_DISCONNECT_EVT

struct esp_ble_gatts_cb_param_t::gatts_open_evt_param open
    Gatt server callback param of ESP_GATTS_OPEN_EVT

struct esp_ble_gatts_cb_param_t::gatts_cancel_open_evt_param cancel_open
    Gatt server callback param of ESP_GATTSCancelar_OPEN_EVT

struct esp_ble_gatts_cb_param_t::gatts_close_evt_param close
    Gatt server callback param of ESP_GATTS_CLOSE_EVT

struct esp_ble_gatts_cb_param_t::gatts_congest_evt_param congest
    Gatt server callback param of ESP_GATTS_CONGEST_EVT

struct esp_ble_gatts_cb_param_t::gatts_rsp_evt_param rsp
    Gatt server callback param of ESP_GATTS_RESPONSE_EVT

struct esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param add_attr_tab
    Gatt server callback param of ESP_GATTS_CREAT_ATTR_TAB_EVT

struct esp_ble_gatts_cb_param_t::gatts_set_attr_val_evt_param set_attr_val
    Gatt server callback param of ESP_GATTS_SET_ATTR_VAL_EVT

struct esp_ble_gatts_cb_param_t::gatts_send_service_change_evt_param service_change
    Gatt server callback param of ESP_GATTS_SEND_SERVICE_CHANGE_EVT

struct gatts_add_attr_tab_evt_param
    #include <esp_gatts_api.h> ESP_GATTS_CREAT_ATTR_TAB_EVT.
**Public Members**

*esp_gatt_status_t* `status`
Operation status

*esp_bt_uuid_t* `svc_uuid`
Service uuid type

`uint8_t` `svc_inst_id`
Service id

`uint16_t` `num_handle`
The number of the attribute handle to be added to the gatts database

`uint16_t` *`handles`
The number to the handles

**struct** `gatts_add_char_descr_evt_param`  
#include `<esp_gatts_api.h>` ESP_GATTS_ADD_CHAR_DESCR_EVT.

**Public Members**

*esp_gatt_status_t* `status`
Operation status

`uint16_t` `attr_handle`
Descriptor attribute handle

`uint16_t` `service_handle`
Service attribute handle

*esp_bt_uuid_t* `descr_uuid`
Characteristic descriptor uuid

**struct** `gatts_add_char_evt_param`  
#include `<esp_gatts_api.h>` ESP_GATTS_ADD_CHAR_EVT.

**Public Members**

*esp_gatt_status_t* `status`
Operation status

`uint16_t` `attr_handle`
Characteristic attribute handle

`uint16_t` `service_handle`
Service attribute handle
Chapter 2. API 参考

### esp bt uuid t char uuid
Characteristic uuid

#### struct gatts_add_incl_srvc_evt_param
#include <esp_gatts_api.h> ESP_GATTS_ADD_INCL_SRVC_EVT.

#### Public Members

#### esp gatt status t status
Operation status

#### uint16_t attr_handle
Included service attribute handle

#### uint16_t service_handle
Service attribute handle

#### struct gatts_cancel_open_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CANCEL_OPEN_EVT.

#### Public Members

#### esp gatt status t status
Operation status

#### struct gatts_close_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CLOSE_EVT.

#### Public Members

#### esp gatt status t status
Operation status

#### uint16_t conn_id
Connection id

#### struct gatts_conf_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CONF_EVT.

#### Public Members

#### esp gatt status t status
Operation status

#### uint16_t conn_id
Connection id
uint16_t handle
attribute handle

uint16_t len
The indication or notification value length, len is valid when send notification or indication failed

uint8_t *value
The indication or notification value, value is valid when send notification or indication failed

struct gatts_congest_evt_param
#include <esp_gatts_api.h> ESP_GATTS_LISTEN_EVT.
ESP_GATTS_CONGEST_EVT

Public Members

uint16_t conn_id
Connection id

bool congested
Congested or not

struct gatts_connect_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CONNECT_EVT.

Public Members

uint16_t conn_id
Connection id

uint8_t link_role
Link role: master role = 0; slave role = 1

esp_bd_addr_t remote_bda
Remote bluetooth device address

esp_gatt_conn_params_t conn_params
current Connection parameters

struct gatts_create_evt_param
#include <esp_gatts_api.h> ESP_GATTS_UNREG_EVT.
ESP_GATTS_CREATE_EVT

Public Members

esp_gatt_status_t status
Operation status
uint16_t service_handle
Service attribute handle

esp_gatt_srvc_id_t service_id
Service id, include service uuid and other information

struct gatts_delete_evt_param
#include <esp_gatts_api.h> ESP_GATTS_DELETE_EVT.

Public Members

esp_gatt_status_t status
Operation status

uint16_t service_handle
Service attribute handle

struct gatts_disconnect_evt_param
#include <esp_gatts_api.h> ESP_GATTS_DISCONNECT_EVT.

Public Members

uint16_t conn_id
Connection id

esp_bd_addr_t remote_bda
Remote bluetooth device address

esp_gatt_conn_reason_t reason
Indicate the reason of disconnection

struct gatts_exec_write_evt_param
#include <esp_gatts_api.h> ESP_GATTS_EXEC_WRITE_EVT.

Public Members

uint16_t conn_id
Connection id

uint32_t trans_id
Transfer id

esp_bd_addr_t bda
The bluetooth device address which been written
uint8_t exec_write_flag
   Execute write flag

struct gatts_mtu_evt_param
   #include <esp_gatts_api.h> ESP_GATTS_MTU_EVT.

Public Members

uint16_t conn_id
   Connection id

uint16_t mtu
   MTU size

struct gatts_open_evt_param
   #include <esp_gatts_api.h> ESP_GATTS_OPEN_EVT.

Public Members

esp_gatt_status_t status
   Operation status

struct gatts_read_evt_param
   #include <esp_gatts_api.h> ESP_GATTS_READ_EVT.

Public Members

uint16_t conn_id
   Connection id

uint32_t trans_id
   Transfer id

esp_bd_addr_t bda
   The bluetooth device address which been read

uint16_t handle
   The attribute handle

uint16_t offset
   Offset of the value, if the value is too long

bool is_long
   The value is too long or not
bool need_rsp
  The read operation need to do response

struct gatts_reg_evt_param
#include <esp_gatts_api.h> ESP_GATTS_REG_EVT.

Public Members

  esp_gatt_status_t status
    Operation status

  uint16_t app_id
    Application id which input in register API

struct gatts_rsp_evt_param
#include <esp_gatts_api.h> ESP_GATTS_RESPONSE_EVT.

Public Members

  esp_gatt_status_t status
    Operation status

  uint16_t handle
    Attribute handle which send response

struct gatts_send_service_change_evt_param
#include <esp_gatts_api.h> ESP_GATTS_SEND_SERVICE_CHANGE_EVT.

Public Members

  esp_gatt_status_t status
    Operation status

struct gatts_set_attr_val_evt_param
#include <esp_gatts_api.h> ESP_GATTS_SET_ATTR_VAL_EVT.

Public Members

  uint16_t srvc_handle
    The service handle

  uint16_t attr_handle
    The attribute handle

  esp_gatt_status_t status
    Operation status
struct gatts_start_evt_param
    #include <esp_gatts_api.h> ESP_GATTS_START_EVT.

Public Members

esp_gatt_status_t status
    Operation status

uint16_t service_handle
    Service attribute handle

struct gatts_stop_evt_param
    #include <esp_gatts_api.h> ESP_GATTS_STOP_EVT.

Public Members

esp_gatt_status_t status
    Operation status

uint16_t service_handle
    Service attribute handle

struct gatts_write_evt_param
    #include <esp_gatts_api.h> ESP_GATTS_WRITE_EVT.

Public Members

uint16_t conn_id
    Connection id

uint32_t trans_id
    Transfer id

esp_bd_addr_t bda
    The bluetooth device address which been written

uint16_t handle
    The attribute handle

uint16_t offset
    Offset of the value, if the value is too long

bool need_rsp
    The write operation need to do response
bool is_prep
    This write operation is prepare write

uint16_t len
    The write attribute value length

uint8_t *value
    The write attribute value

Macros

ESP_GATT_PREP_WRITE_CANCEL
    Prepare write flag to indicate cancel prepare write

ESP_GATT_PREP_WRITE_EXEC
    Prepare write flag to indicate execute prepare write

Type Definitions

typedef void (*esp_gatts_cb_t)(esp_gatts_cb_event_t event, esp_gatt_if_t gatts_if, esp_ble_gatts_cb_param_t *param)
    GATT Server callback function type.

    Param event : Event type
    Param gatts_if : GATT server access interface, normally different gatts_if correspond to different profile
    Param param : Point to callback parameter, currently is union type

Enumerations

enum esp_gatts_cb_event_t
    GATT Server callback function events.

    Values:

    enumerator ESP_GATTS_REG_EVT
        When register application id, the event comes

    enumerator ESP_GATTS_READ_EVT
        When gatt client request read operation, the event comes

    enumerator ESP_GATTS_WRITE_EVT
        When gatt client request write operation, the event comes

    enumerator ESP_GATTS_EXEC_WRITE_EVT
        When gatt client request execute write, the event comes

    enumerator ESP_GATTS_MTU_EVT
        When set mtu complete, the event comes

    enumerator ESP_GATTS_CONF_EVT
        When receive confirm, the event comes
enumerator **ESP_GATTS_UNREG_EVT**
   When unregister application id, the event comes

enumerator **ESP_GATTS_CREATE_EVT**
   When create service complete, the event comes

enumerator **ESP_GATTS_ADD_INCL_SRVC_EVT**
   When add included service complete, the event comes

enumerator **ESP_GATTS_ADD_CHAR_EVT**
   When add characteristic complete, the event comes

enumerator **ESP_GATTS_ADD_CHAR_DESCR_EVT**
   When add descriptor complete, the event comes

enumerator **ESP_GATTS_DELETE_EVT**
   When delete service complete, the event comes

enumerator **ESP_GATTS_START_EVT**
   When start service complete, the event comes

enumerator **ESP_GATTS_STOP_EVT**
   When stop service complete, the event comes

enumerator **ESP_GATTS_CONNECT_EVT**
   When gatt client connect, the event comes

enumerator **ESP_GATTS_DISCONNECT_EVT**
   When gatt client disconnect, the event comes

enumerator **ESP_GATTS_OPEN_EVT**
   When connect to peer, the event comes

enumerator **ESP_GATTS_CANCEL_OPEN_EVT**
   When disconnect from peer, the event comes

enumerator **ESP_GATTS_CLOSE_EVT**
   When gatt server close, the event comes

enumerator **ESP_GATTS_LISTEN_EVT**
   When gatt listen to be connected the event comes

enumerator **ESP_GATTS_CONGEST_EVT**
   When congest happen, the event comes

enumerator **ESP_GATTS_RESPONSE_EVT**
   When gatt send response complete, the event comes
enumerator **ESP_GATTS_CREAT_ATTR_TAB_EVT**  
When gatt create table complete, the event comes

enumerator **ESP_GATTS_SET_ATTR_VAL_EVT**  
When gatt set attr value complete, the event comes

enumerator **ESP_GATTS_SEND_SERVICE_CHANGE_EVT**  
When gatt send service change indication complete, the event comes

**GATT CLIENT API**

**Application Example**  
Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a GATT client demo and its tutorial. This demo can scan for devices, connect to the GATT server and discover its services.
  - bluetooth/bluedroid/ble/gatt_client
  - GATT Client Example Walkthrough

- This is a multiple connection demo and its tutorial. This demo can connect to multiple GATT server devices and discover their services.
  - bluetooth/bluedroid/ble/gattc_multi_connect
  - GATT Client Multi-connection Example Walkthrough

- This is a BLE SPP-Like demo. This demo, which acts as a GATT client, can receive data from UART and then send the data to the peer device automatically.
  - bluetooth/bluedroid/ble/ble_spp_client

**API Reference**

**Header File**

- components/bt/host/bluedroid/api/include/api/esp_gattc_api.h

**Functions**

- **esp_err_t esp_ble_gattc_register_callback (esp_gattc_cb_t callback)**  
  This function is called to register application callbacks with GATTC module.
  
  参数 **callback** - [in]: pointer to the application callback function.
  
  返回
  - ESP_OK: success
  - other: failed

- **esp_err_t esp_ble_gattc_app_register (uint16_t app_id)**  
  This function is called to register application callbacks with GATTC module.
  
  参数 **app_id** - [in]: Application Identify (UUID), for different application
  
  返回
  - ESP_OK: success
  - other: failed

- **esp_err_t esp_ble_gattc_app_unregister (esp_gatt_if_t gattc_if)**  
  This function is called to unregister an application from GATTC module.
  
  参数 **gattc_if** - [in]: Gatt client access interface.
  
  返回
  - ESP_OK: success
  - other: failed
esp_err_t esp_ble_gattc_open(esp_gatt_if_t gattc_if, esp_bd_addr_t remote_bda, esp_ble_addr_type_t remote_addr_type, bool is_direct)

Open a direct connection or add a background auto connection.

参数
- remote_bda [in] remote device bluetooth device address.
- remote_addr_type [in] remote device bluetooth device the address type.
- is_direct [in] direct connection or background auto connection(by now, background auto connection is not supported).

返回
- ESP_OK: success
- other: failed

esp_err_t esp_ble_gattc_aux_open(esp_gatt_if_t gattc_if, esp_bd_addr_t remote_bda, esp_ble_addr_type_t remote_addr_type, bool is_direct)

esp_err_t esp_ble_gattc_close(esp_gatt_if_t gattc_if, uint16_t conn_id)

Close the virtual connection to the GATT server. gattc may have multiple virtual GATT server connections when multiple app_id registered, this API only close one virtual GATT server connection. if there exist other virtual GATT server connections, it does not disconnect the physical connection. if you want to disconnect the physical connection directly, you can use esp_ble_gap_disconnect(esp_bd_addr_t remote_device).

参数
- conn_id [in] connection ID to be closed.

返回
- ESP_OK: success
- other: failed

esp_err_t esp_ble_gattc_send_mtu_req(esp_gatt_if_t gattc_if, uint16_t conn_id)

Configure the MTU size in the GATT channel. This can be done only once per connection. Before using, use esp_ble_gatt_set_local_mtu() to configure the local MTU size.

参数
- conn_id [in] connection ID.

返回
- ESP_OK: success
- other: failed

esp_err_t esp_ble_gattc_search_service(esp_gatt_if_t gattc_if, uint16_t conn_id, esp_bt_uuid_t *filter_uuid)

This function is called to get service from local cache. This function report service search result by a callback event, and followed by a service search complete event.

参数
- conn_id [in] connection ID.
- filter_uuid [in] a UUID of the service application is interested in. If Null, discover for all services.

返回
- ESP_OK: success
- other: failed

esp_gatt_status_t esp_ble_gattc_get_service(esp_gatt_if_t gattc_if, uint16_t conn_id, esp_bt_uuid_t *filter_uuid, esp_gattc_service_elem_t *result, uint16_t *count)

Find all the service with the given service uuid in the gattc cache, if the svc_uuid is NULL, find all the service. Note: It just get service from local cache, won’t get from remote devices. If want to get it from remote device, need to used the esp_ble_gattc_cache_refresh, then call esp_ble_gattc_get_service again.
参数
• `gattc_if` - [in] Gatt client access interface.
• `conn_id` - [in] connection ID which identify the server.
• `svc_uuid` - [in] the pointer to the service uuid.
• `result` - [out] The pointer to the service which has been found in the gattc cache.
• `count` - [inout] input the number of service want to find, it will output the number of service has been found in the gattc cache with the given service uuid.
• `offset` - [in] Offset of the service position to get.

返回
• ESP_OK: success
• other: failed

```
esp_gatt_status_t esp_ble_gattc_get_all_char(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t
start_handle, uint16_t end_handle,
esp_gattc_char_elem_t *result, uint16_t *count, uint16_t
offset)
```

Find all the characteristic with the given service in the gattc cache Note: It just get characteristic from local cache, won’t get from remote devices.

参数
• `gattc_if` - [in] Gatt client access interface.
• `conn_id` - [in] connection ID which identify the server.
• `start_handle` - [in] the attribute start handle.
• `end_handle` - [in] the attribute end handle.
• `result` - [out] The pointer to the characteristic in the service.
• `count` - [inout] input the number of characteristic want to find, it will output the number of characteristic has been found in the gattc cache with the given service.
• `offset` - [in] Offset of the characteristic position to get.

返回
• ESP_OK: success
• other: failed

```
esp_gatt_status_t esp_ble_gattc_get_all_descr(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t
char_handle, esp_gattc_descr_elem_t *result, uint16_t *count, uint16_t
offset)
```

Find all the descriptor with the given characteristic in the gattc cache Note: It just get descriptor from local cache, won’t get from remote devices.

参数
• `gattc_if` - [in] Gatt client access interface.
• `conn_id` - [in] connection ID which identify the server.
• `char_handle` - [in] the given characteristic handle
• `result` - [out] The pointer to the descriptor in the characteristic.
• `count` - [inout] input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.
• `offset` - [in] Offset of the descriptor position to get.

返回
• ESP_OK: success
• other: failed

```
esp_gatt_status_t esp_ble_gattc_get_char_by_uuid(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t
start_handle, uint16_t end_handle, esp_bt_uuid_t
char_uuid, esp_gattc_char_elem_t *result,
uint16_t *count)
```

Find the characteristic with the given characteristic uuid in the gattc cache Note: It just get characteristic from local cache, won’t get from remote devices.

参数
• `gattc_if` - [in] Gatt client access interface.
• `conn_id` - [in] connection ID which identify the server.
Chapter 2. API

- **start_handle**  [in]  the attribute start handle
- **end_handle**  [in]  the attribute end handle
- **char_uuid**  [in]  the characteristic uuid
- **result**  [out]  The pointer to the characteristic in the service.
- **count**  [inout]  input the number of characteristic want to find, it will output the number of characteristic has been found in the gattc cache with the given service.

返回
- ESP_OK: success
- other: failed

```c
esp_gatt_status_t esp_ble_gattc_get_descr_by_uuid(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, esp_btUuid_t char_uuid, esp_btUuid_t descr_uuid, esp_gattc_descr_elem_t *result, uint16_t *count)
```

Find the descriptor with the given characteristic uuid in the gattc cache Note: It just get descriptor from local cache, won’t get from remote devices.

参数
- **gattc_if**  [in]  Gatt client access interface.
- **conn_id**  [in]  connection ID which identify the server.
- **start_handle**  [in]  the attribute start handle
- **end_handle**  [in]  the attribute end handle
- **char_uuid**  [in]  the characteristic uuid.
- **descr_uuid**  [in]  the descriptor uuid.
- **result**  [out]  The pointer to the descriptor in the given characteristic.
- **count**  [inout]  input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.

返回
- ESP_OK: success
- other: failed

```c
esp_gatt_status_t esp_ble_gattc_get_descr_by_char_handle(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t char_handle, esp_btUuid_t descr_uuid, esp_gattc_descr_elem_t *result, uint16_t *count)
```

Find the descriptor with the given characteristic handle in the gattc cache Note: It just get descriptor from local cache, won’t get from remote devices.

参数
- **gattc_if**  [in]  Gatt client access interface.
- **conn_id**  [in]  connection ID which identify the server.
- **char_handle**  [in]  the characteristic handle.
- **descr_uuid**  [in]  the descriptor uuid.
- **result**  [out]  The pointer to the descriptor in the given characteristic.
- **count**  [inout]  input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.

返回
- ESP_OK: success
- other: failed

```c
esp_gatt_status_t esp_ble_gattc_get_include_service(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, esp_btUuid_t *incl_uuid, esp_gattc_incl_svc_elem_t *result, uint16_t *count)
```

Find the include service with the given service handle in the gattc cache Note: It just get include service from local cache, won’t get from remote devices.

参数
• gattc_if [in] Gatt client access interface.
• conn_id [in] connection ID which identify the server.
• start_handle [in] the attribute start handle
• end_handle [in] the attribute end handle
• incl_uuid [in] the include service uuid
• result [out] The pointer to the include service in the given service.
• count [inout] input the number of include service want to find, it will output the number of include service has been found in the gattc cache with the given service.

返回
• ESP_OK: success
• other: failed

ESP_OK 出现：success
其他：failed

esp_gatt_status_t esp_ble_gattc_get_attr_count (esp_gatt_if_t gattc_if, uint16_t conn_id, esp_gatt_db_attr_type_t type, uint16_t start_handle, uint16_t end_handle, uint16_t char_handle, uint16_t *count)

Find the attribute count with the given service or characteristic in the gattc cache.

参数
• gattc_if [in] Gatt client access interface.
• conn_id [in] connection ID which identify the server.
• type [in] the attribute type.
• start_handle [in] the attribute start handle, if the type is ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore.
• end_handle [in] the attribute end handle, if the type is ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore.
• char_handle [in] the characteristic handle, this parameter valid when the type is ESP_GATT_DB_DESCRIPTOR. If the type isn’t ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore.
• count [out] output the number of attribute has been found in the gattc cache with the given attribute type.

返回
• ESP_OK: success
• other: failed

ESP_OK 出现：success
其他：failed

esp_gatt_status_t esp_ble_gattc_get_db (esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, esp_gattc_db_elem_t *db, uint16_t *count)

This function is called to get the GATT database. Note: It just get attribute data base from local cache, won’t get from remote devices.

参数
• gattc_if [in] Gatt client access interface.
• start_handle [in] the attribute start handle
• end_handle [in] the attribute end handle
• conn_id [in] connection ID which identify the server.
• db [in] output parameter which will contain the GATT database copy. Caller is responsible for freeing it.
• count [in] number of elements in database.

返回
• ESP_OK: success
• other: failed

ESP_OK 出现：success
其他：failed

esp_err_t esp_ble_gattc_read_char (esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t handle, esp_gatt_auth_req_t auth_req)

This function is called to read a service’s characteristics of the given characteristic handle.

参数
• gattc_if [in] Gatt client access interface.
• conn_id [in]: connection ID.
• handle [in]: characteristic handle to read.
**Chapter 2. API**

- **auth_req** - [in]: authenticate request type
  - ESP_OK: success
  - other: failed

**Esp_err_t esp_ble_gattc_read_by_type** (esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t start_handle, uint16_t end_handle, esp_bt_uuid_t *uuid, esp_gatt_auth_req_t auth_req)

This function is called to read a service’s characteristics of the given characteristic UUID.

**参数**
- gattc_if - [in]: Gatt client access interface.
- conn_id - [in]: connection ID.
- start_handle - [in]: the attribute start handle.
- end_handle - [in]: the attribute end handle
- uuid - [in]: The UUID of attribute which will be read.
- auth_req - [in]: authenticate request type

**返回**
- ESP_OK: success
- other: failed

**Esp_err_t esp_ble_gattc_read_multiple** (esp_gatt_if_t gattc_if, uint16_t conn_id, esp_gattc_multi_t *read_multi, esp_gatt_auth_req_t auth_req)

This function is called to read multiple characteristic or characteristic descriptors.

**参数**
- gattc_if - [in]: Gatt client access interface.
- conn_id - [in]: connection ID.
- read_multi - [in]: pointer to the read multiple parameter.
- auth_req - [in]: authenticate request type

**返回**
- ESP_OK: success
- other: failed

**Esp_err_t esp_ble_gattc_read_char_descr** (esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t handle, esp_gatt_auth_req_t auth_req)

This function is called to read a characteristics descriptor.

**参数**
- gattc_if - [in]: Gatt client access interface.
- conn_id - [in]: connection ID.
- handle - [in]: descriptor handle to read.
- auth_req - [in]: authenticate request type

**返回**
- ESP_OK: success
- other: failed

**Esp_err_t esp_ble_gattc_write_char** (esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t handle, uint16_t value_len, uint8_t *value, esp_gatt_write_type_t write_type, esp_gatt_auth_req_t auth_req)

This function is called to write characteristic value.

**参数**
- gattc_if - [in]: Gatt client access interface.
- conn_id - [in]: connection ID.
- handle - [in]: characteristic handle to write.
- value_len - [in]: length of the value to be written.
- value - [in]: the value to be written.
- write_type - [in]: the type of attribute write operation.
- auth_req - [in]: authentication request.

**返回**
• ESP_OK: success
• other: failed

```c
esp_err_t esp_ble_gattc_write_char_descr(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t handle, uint16_t value_len, uint8_t *value, esp_gatt_write_type_t write_type, esp_gatt_auth_req_t auth_req)
```

This function is called to write characteristic descriptor value.

参数
- conn_id - [in] : connection ID
- handle - [in] : descriptor handle to write.
- value_len - [in] length of the value to be written.
- value - [in] : the value to be written.
- write_type - [in] : the type of attribute write operation.

返回
- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gattc_prepare_write(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t handle, uint16_t offset, uint16_t value_len, uint8_t *value, esp_gatt_auth_req_t auth_req)
```

This function is called to prepare write a characteristic value.

参数
- conn_id - [in] : connection ID.
- handle - [in] : characteristic handle to prepare write.
- offset - [in] : offset of the write value.
- value_len - [in] length of the value to be written.
- value - [in] : the value to be written.

返回
- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gattc_prepare_write_char_descr(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t handle, uint16_t offset, uint16_t value_len, uint8_t *value, esp_gatt_auth_req_t auth_req)
```

This function is called to prepare write a characteristic descriptor value.

参数
- conn_id - [in] : connection ID.
- handle - [in] : characteristic descriptor handle to prepare write.
- offset - [in] : offset of the write value.
- value_len - [in] length of the value to be written.
- value - [in] : the value to be written.

返回
- ESP_OK: success
- other: failed

```c
esp_err_t esp_ble_gattc_execute_write(esp_gatt_if_t gattc_if, uint16_t conn_id, bool is_execute)
```

This function is called to execute write a prepare write sequence.

参数
- conn_id - [in] : connection ID.
API

- **is_execute** - [in] execute or cancel.
  - ESP_OK: success
  - other: failed

`esp_err_t esp_ble_gattc_register_for_notify (esp_gatt_if_t gattc_if, esp_bd_addr_t server_bda, uint16_t handle)`

This function is called to register for notification of a service.

- **server_bda** - [in]: target GATT server.
- **handle** - [in]: GATT characteristic handle.

- **ESP_OK**: registration succeeds
- **other**: failed

`esp_err_t esp_ble_gattc_unregister_for_notify (esp_gatt_if_t gattc_if, esp_bd_addr_t server_bda, uint16_t handle)`

This function is called to de-register for notification of a service.

- **server_bda** - [in]: target GATT server.
- **handle** - [in]: GATT characteristic handle.

- **ESP_OK**: unregistration succeeds
- **other**: failed

`esp_err_t esp_ble_gattc_cache_refresh (esp_bd_addr_t remote_bda)`

Refresh the server cache store in the gattc stack of the remote device. If the device is connected, this API will restart the discovery of service information of the remote device.

- **remote_bda** - [in] remote device BD address.

- **ESP_OK**: success
- **other**: failed

`esp_err_t esp_ble_gattc_cache_assoc (esp_gatt_if_t gattc_if, esp_bd_addr_t src_addr, esp_bd_addr_t assoc_addr, bool is_assoc)`

Add or delete the associated address with the source address. Note: The role of this API is mainly when the client side has stored a server-side database, when it needs to connect another device, but the device’s attribute database is the same as the server database stored on the client-side, calling this API can use the database that the device has stored used as the peer server database to reduce the attribute database search and discovery process and speed up the connection time. The associated address mains that device want to used the database that has stored in the local cache. The source address mains that device want to share the database to the associated address device.

- **src_addr** - [in] the source address which provide the attribute table.
- **assoc_addr** - [in] the associated device address which went to share the attribute table with the source address.
- **is_assoc** - [in] true add the associated device address, false remove the associated device address.

- **ESP_OK**: success
- **other**: failed

`esp_err_t esp_ble_gattc_cache_get_addr_list (esp_gatt_if_t gattc_if)`

Get the address list which has store the attribute table in the gattc cache. There will callback
ESP_GATTC_GET_ADDR_LIST_EVT event when get address list complete.


返回
• ESP_OK: success
• other: failed

esp_err_t esp_ble_gattc_cache_clean (esp_bd_addr_t remote_bda)
Clean the service cache of this device in the gattc stack.

参数 remote_bda [in] remote device BD address.

返回
• ESP_OK: success
• other: failed

Unions

union esp_ble_gattc_cb_param_t

#include <esp_gattc_api.h> Gatt client callback parameters union.

Public Members

struct esp_ble_gattc_cb_param_t::gattc_reg_evt_param reg
Gatt client callback param of ESP_GATTC_REG_EVT

struct esp_ble_gattc_cb_param_t::gattc_open_evt_param open
Gatt client callback param of ESP_GATTC_OPEN_EVT

struct esp_ble_gattc_cb_param_t::gattc_close_evt_param close
Gatt client callback param of ESP_GATTC_CLOSE_EVT

struct esp_ble_gattc_cb_param_t::gattc_cfg_mtu_evt_param cfg_mtu
Gatt client callback param of ESP_GATTC_CFG_MTU_EVT

struct esp_ble_gattc_cb_param_t::gattc_search_cmpl_evt_param search_cmpl
Gatt client callback param of ESP_GATTC_SEARCH_CMPL_EVT

struct esp_ble_gattc_cb_param_t::gattc_search_res_evt_param search_res
Gatt client callback param of ESP_GATTC_SEARCH_RES_EVT

struct esp_ble_gattc_cb_param_t::gattc_read_char_evt_param read
Gatt client callback param of ESP_GATTC_READ_CHAR_EVT

struct esp_ble_gattc_cb_param_t::gattc_write_evt_param write
Gatt client callback param of ESP_GATTC_WRITE_DESCR_EVT

struct esp_ble_gattc_cb_param_t::gattc_exec_cmpl_evt_param exec_cmpl
Gatt client callback param of ESP_GATTC_EXEC_EVT

struct esp_ble_gattc_cb_param_t::gattc_notify_evt_param notify
Gatt client callback param of ESP_GATTC_NOTIFY_EVT
struct esp_ble_gattc_cb_param_t::gattc_srvc_chg_evt_param srvc_chg
    Gatt client callback param of ESP_GATTC_SRVC_CHG_EVT

struct esp_ble_gattc_cb_param_t::gattc_congest_evt_param congest
    Gatt client callback param of ESP_GATTC_CONGEST_EVT

struct esp_ble_gattc_cb_param_t::gattc_reg_for_notify_evt_param reg_for_notify
    Gatt client callback param of ESP_GATTC_REG_FOR_NOTIFY_EVT

struct esp_ble_gattc_cb_param_t::gattc_unreg_for_notify_evt_param unreg_for_notify
    Gatt client callback param of ESP_GATTC_UNREG_FOR_NOTIFY_EVT

struct esp_ble_gattc_cb_param_t::gattc_connect_evt_param connect
    Gatt client callback param of ESP_GATTC_CONNECT_EVT

struct esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param disconnect
    Gatt client callback param of ESP_GATTC_DISCONNECT_EVT

struct esp_ble_gattc_cb_param_t::gattc_set_assoc_addr_cmp_evt_param set_assoc_cmp
    Gatt client callback param of ESP_GATTC_SET_ASSOC_EVT

struct esp_ble_gattc_cb_param_t::gattc_get_addr_list_evt_param get_addr_list
    Gatt client callback param of ESP_GATTC_GET_ADDR_LIST_EVT

struct esp_ble_gattc_cb_param_t::gattc_queue_full_evt_param queue_full
    Gatt client callback param of ESP_GATTC_QUEUE_FULL_EVT

struct esp_ble_gattc_cb_param_t::gattc_dis_srvc_cmpl_evt_param dis_srvc_cmpl
    Gatt client callback param of ESP_GATTC_DIS_SRVC_CMPL_EVT

struct gattc_cfg_mtu_evt_param
    #include <esp_gattc_api.h> ESP_GATTC_CFG_MTU_EVT.

**Public Members**

* esp_gatt_status_t status
  Operation status

* uint16_t conn_id
  Connection id

* uint16_t mtu
  MTU size

struct gattc_close_evt_param
    #include <esp_gattc_api.h> ESP_GATTC_CLOSE_EVT.
### Public Members

```c
esp_gatt_status_t status
```
Operation status

```c
uint16_t conn_id
```
Connection id

```c
esp_bd_addr_t remote_bda
```
Remote bluetooth device address

```c
esp_gatt_conn_reason_t reason
```
The reason of gatt connection close

```c
struct gattc_congest_evt_param
#include <esp_gattc_api.h> ESP_GATTC_CONGEST_EVT.
```

### Public Members

```c
uint16_t conn_id
```
Connection id

```c
bool congested
```
Congested or not

```c
struct gattc_connect_evt_param
#include <esp_gattc_api.h> ESP_GATTC_CONNECT_EVT.
```

### Public Members

```c
uint16_t conn_id
```
Connection id

```c
uint8_t link_role
```
Link role: master role = 0; slave role = 1

```c
esp_bd_addr_t remote_bda
```
Remote bluetooth device address

```c
esp_gatt_conn_params_t conn_params
```
current connection parameters

```c
struct gattc_dis_srvc_cmpl_evt_param
#include <esp_gattc_api.h> ESP_GATTC_DIS_SRVCCMPL_EVT.
```
**Public Members**

`esp_gatt_status_t status`  
Operation status

`uint16_t conn_id`  
Connection id

```c
struct gattc_disconnect_evt_param
#include <esp_gattc_api.h> ESP_GATTC_DISCONNECT_EVT.
```

**Public Members**

`esp_gatt_conn_reason_t reason`  
Disconnection reason

`uint16_t conn_id`  
Connection id

```c
esp_bd_addr_t remote_bda
Remote bluetooth device address
```

```c
struct gattc_exec_cmpl_evt_param
#include <esp_gattc_api.h> ESP_GATTC_EXEC_EVT.
```

**Public Members**

`esp_gatt_status_t status`  
Operation status

`uint16_t conn_id`  
Connection id

```c
struct gattc_get_addr_list_evt_param
#include <esp_gattc_api.h> ESP_GATTC_GET_ADDR_LIST_EVT.
```

**Public Members**

`esp_gatt_status_t status`  
Operation status

`uint8_t num_addr`  
The number of address in the gattc cache address list

```c
esp_bd_addr_t *addr_list
The pointer to the address list which has been get from the gattc cache
```
struct gattc_notify_evt_param
#include <esp_gattc_api.h> ESP_GATTC_NOTIFY_EVT.

Public Members

uint16_t conn_id
Connection id

esp_bd_addr_t remote_bda
Remote bluetooth device address

uint16_t handle
The Characteristic or descriptor handle

uint16_t value_len
Notify attribute value

uint8_t* value
Notify attribute value

bool is_notify
True means notify, false means indicate

struct gattc_open_evt_param
#include <esp_gattc_api.h> ESP_GATTC_OPEN_EVT.

Public Members

esp_gatt_status_t status
Operation status

uint16_t conn_id
Connection id

esp_bd_addr_t remote_bda
Remote bluetooth device address

uint16_t mtu
MTU size

struct gattc_queue_full_evt_param
#include <esp_gattc_api.h> ESP_GATTC_QUEUE_FULL_EVT.

Public Members


```c
struct gattc_read_char_evt_param
#include <esp_gattc_api.h> ESP_GATTC_READ_CHAR_EVT, ESP_GATTC_READ_DESCR_EVT.
```

**Public Members**

```c
esp_gatt_status_t status
Operation status
```

```c
uint16_t conn_id
Connection id
```

```c
bool is_full
The gattc command queue is full or not
```

```c
struct gattc_reg_evt_param
#include <esp_gattc_api.h> ESP_GATTC_REG_EVT.
```

**Public Members**

```c
esp_gatt_status_t status
Operation status
```

```c
uint16_t conn_id
Connection id
```

```c
uint16_t handle
Characteristic handle
```

```c
uint8_t *value
Characteristic value
```

```c
uint16_t value_len
Characteristic value length
```

```c
struct gattc_reg_for_notify_evt_param
#include <esp_gattc_api.h> ESP_GATTC_REG_FOR_NOTIFY_EVT.
```

**Public Members**

```c
esp_gatt_status_t status
Operation status
```

```c
uint16_t app_id
Application id which input in register API
```

```c
struct gattc_reg_for_notify_evt_param
#include <esp_gattc_api.h> ESP_GATTC_REG_FOR_NOTIFY_EVT.
```

**Public Members**
Chapter 2. API

**Public Members**

```
esp_gatt_status_t status
Operation status
```

```
uint16_t handle
The characteristic or descriptor handle
```

```
struct gattc_search_cmpl_evt_param
#include <esp_gattc_api.h> ESP_GATTC_SEARCH_CMPL_EVT.
```

```
public Members

esp_gatt_status_t status
Operation status
```

```
uint16_t conn_id
Connection id
```

```
esp_service_source_t searched_service_source
The source of the service information
```

```
struct gattc_search_res_evt_param
#include <esp_gattc_api.h> ESP_GATTC_SEARCH_RES_EVT.
```

```
Public Members

uint16_t conn_id
Connection id
```

```
uint16_t start_handle
Service start handle
```

```
uint16_t end_handle
Service end handle
```

```
esp_gatt_id_t srcvc_id
Service id, include service uuid and other information
```

```bool is_primary
True if this is the primary service
```

```
struct gattc_set_assoc_addr_cmp_evt_param
#include <esp_gattc_api.h> ESP_GATTC_SET_ASSOC_EVT.
```

```
Public Members
```
### Chapter 2. API Reference

#### esp_gatt_status_t status
- Operation status

#### struct gattc_svc_chg_evt_param
- #include <esp_gattc_api.h> ESP_GATTC_SRVC_CHG_EVT.

**Public Members**

- esp_bd_addr_t remote_bda
  - Remote bluetooth device address

#### struct gattc_unreg_for_notify_evt_param
- #include <esp_gattc_api.h> ESP_GATTC_UNREG_FOR_NOTIFY_EVT.

**Public Members**

- esp_gatt_status_t status
  - Operation status

- uint16_t handle
  - The characteristic or descriptor handle

#### struct gattc_write_evt_param
- #include <esp_gattc_api.h> ESP_GATTC_WRITE_CHAR_EVT, ESP_GATTC_PREP_WRITE_EVT, ESP_GATTC_WRITE_DESCR_EVT.

**Public Members**

- esp_gatt_status_t status
  - Operation status

- uint16_t conn_id
  - Connection id

- uint16_t handle
  - The characteristic or descriptor handle

- uint16_t offset
  - The prepare write offset, this value is valid only when prepare write

#### Type Definitions

typedef void (*esp_gattc_cb_t)(esp_gattc_cb_event_t event, esp_gatt_if_t gattc_if, esp_ble_gattc_cb_param_t *param)

  GATT Client callback function type.

  **Param event** : Event type
**Param gattc_if** : GATT client access interface, normally different gattc_if correspond to different profile

**Param param** : Point to callback parameter, currently is union type

### Enumerations

```c
enum esp_gattc_cb_event_t
{
    GATTClientcallbackfunctionevents.

    Values:

    enumerator ESP_GATTC_REG_EVT
        When GATT client is registered, the event comes

    enumerator ESP_GATTC_UNREG_EVT
        When GATT client is unregistered, the event comes

    enumerator ESP_GATTC_OPEN_EVT
        When GATT virtual connection is set up, the event comes

    enumerator ESP_GATTC_READ_CHAR_EVT
        When GATT characteristic is read, the event comes

    enumerator ESP_GATTC_WRITE_CHAR_EVT
        When GATT characteristic write operation completes, the event comes

    enumerator ESP_GATTC_CLOSE_EVT
        When GATT virtual connection is closed, the event comes

    enumerator ESP_GATTC_SEARCH_CMPL_EVT
        When GATT service discovery is completed, the event comes

    enumerator ESP_GATTC_SEARCH_RES_EVT
        When GATT service discovery result is got, the event comes

    enumerator ESP_GATTC_READ_DESCR_EVT
        When GATT characteristic descriptor read completes, the event comes

    enumerator ESP_GATTC_WRITE_DESCR_EVT
        When GATT characteristic descriptor write completes, the event comes

    enumerator ESP_GATTC_NOTIFY_EVT
        When GATT notification or indication arrives, the event comes

    enumerator ESP_GATTC_PREP_WRITE_EVT
        When GATT prepare-write operation completes, the event comes

    enumerator ESP_GATTC_EXEC_EVT
        When write execution completes, the event comes
```
enumerator **ESP_GATTC_ACL_EVT**
When ACL connection is up, the event comes

enumerator **ESP_GATTC_CANCEL_OPEN_EVT**
When GATT client ongoing connection is cancelled, the event comes

enumerator **ESP_GATTC_SRVC_CHG_EVT**
When "service changed" occurs, the event comes

enumerator **ESP_GATTC_ENC_CMPL_CB_EVT**
When encryption procedure completes, the event comes

enumerator **ESP_GATTC_CFG_MTU_EVT**
When configuration of MTU completes, the event comes

enumerator **ESP_GATTC_ADV_DATA_EVT**
When advertising of data, the event comes

enumerator **ESP_GATTC_MULT_ADV_ENB_EVT**
When multi-advertising is enabled, the event comes

enumerator **ESP_GATTC_MULT_ADV_UPD_EVT**
When multi-advertising parameters are updated, the event comes

enumerator **ESP_GATTC_MULT_ADV_DATA_EVT**
When multi-advertising data arrives, the event comes

enumerator **ESP_GATTC_MULT_ADV_DIS_EVT**
When multi-advertising is disabled, the event comes

enumerator **ESP_GATTC_CONGEST_EVT**
When GATT connection congestion comes, the event comes

enumerator **ESP_GATTC_BTH_SCAN_ENB_EVT**
When batch scan is enabled, the event comes

enumerator **ESP_GATTC_BTH_SCAN_CFG_EVT**
When batch scan storage is configured, the event comes

enumerator **ESP_GATTC_BTH_SCAN_RD_EVT**
When Batch scan read event is reported, the event comes

enumerator **ESP_GATTC_BTH_SCAN_THR_EVT**
When Batch scan threshold is set, the event comes

enumerator **ESP_GATTC_BTH_SCAN_PARAM_EVT**
When Batch scan parameters are set, the event comes
Chapter 2. API

**enumerator** ESP_GATTC_BTH_SCAN_DIS_EVT
When Batch scan is disabled, the event comes

**enumerator** ESP_GATTC_SCAN_FLT_CFG_EVT
When Scan filter configuration completes, the event comes

**enumerator** ESP_GATTC_SCAN_FLT_PARAM_EVT
When Scan filter parameters are set, the event comes

**enumerator** ESP_GATTC_SCAN_FLT_STATUS_EVT
When Scan filter status is reported, the event comes

**enumerator** ESP_GATTC_ADV_VSC_EVT
When advertising vendor spec content event is reported, the event comes

**enumerator** ESP_GATTC_REG_FOR_NOTIFY_EVT
When register for notification of a service completes, the event comes

**enumerator** ESP_GATTC_UNREG_FOR_NOTIFY_EVT
When unregister for notification of a service completes, the event comes

**enumerator** ESP_GATTC_CONNECT_EVT
When the ble physical connection is set up, the event comes

**enumerator** ESP_GATTC_DISCONNECT_EVT
When the ble physical connection disconnected, the event comes

**enumerator** ESP_GATTC_READ_MULTIPLE_EVT
When the ble characteristic or descriptor multiple complete, the event comes

**enumerator** ESP_GATTC_QUEUE_FULL_EVT
When the gattc command queue full, the event comes

**enumerator** ESP_GATTC_SET_ASSOC_EVT
When the ble gattc set the associated address complete, the event comes

**enumerator** ESP_GATTC_GET_ADDR_LIST_EVT
When the ble get gattc address list in cache finish, the event comes

**enumerator** ESP_GATTC_DIS_SRVC_CMPL_EVT
When the ble discover service complete, the event comes

### BLUFI API

**Overview**  
BLUFI is a profile based GATT to config ESP32 WIFI to connect/disconnect AP or setup a softap and etc. Use should concern these things:

1. The event sent from profile. Then you need to do something as the event indicate.
2. Security reference. You can write your own Security functions such as symmetrical encryption/decryption and checksum functions. Even you can define the  “Key Exchange/Negotiation” procedure.
**Application Example** Check bluetooth folder in ESP-IDF examples, which contains the following application:

- This is the BLUFI demo. This demo can set ESP32’s wifi to softap/station/softap&station mode and config wifi connections - bluetooth/blufi

**API Reference**

**Header File**

- components/bt/common/api/include/api/esp_blufi_api.h

**Functions**

```c
#include <api/esp_blufi_api.h>

#define BLUFI_API_API_VERSION_MAJOR 0
#define BLUFI_API_API Version_MINOR 1
#define BLUFI_API_API Version_PATCH 0

// ESP_OK success, other failed

/// 
/// This function is called to receive blufi callback event.
///
/// @param callbacks [in] callback functions
/// @param ESP_OK - success, other - failed

esp_err_t esp_blufi_register_callbacks(esp_blufi_callbacks_t *callbacks)

/// 
/// This function is called to initialize blufi_profile.
///
/// @param ESP_OK - success, other - failed

esp_err_t esp_blufi_profile_init(void)

/// 
/// This function is called to de-initialize blufi_profile.
///
/// @param ESP_OK - success, other - failed

esp_err_t esp_blufi_profile_deinit(void)

/// 
/// This function is called to send wifi connection report.
///
/// @param opmode [in] wifi opmode
/// @param sta_conn_state [in] station is already in connection or not
/// @param softap_conn_num [in] softap connection number
/// @param extra_info [in] extra information, such as sta_ssid, softap_ssid and etc.
/// @param ESP_OK - success, other - failed

esp_err_t esp_blufi_send_wifi_conn_report(wifi_mode_t opmode, esp_blufi_sta_conn_state_t sta_conn_state, uint8_t softap_conn_num, esp_blufi_extra_info_t *extra_info)

/// 
/// This function is called to send wifi list.
///
/// @param apCount [in] wifi list count
/// @param list [in] wifi list
/// @param ESP_OK - success, other - failed

uint16_t esp_blufi_send_wifi_list(uint16_t apCount, esp_blufi_ap_record_t *list)

/// 
/// Get BLUFI profile version.
///
/// @param ESP_OK - success, other - failed

uint16_t esp_blufi_get_version(void)

/// 
/// This function is called to send blufi error information.
///
/// @param state [in] error state
/// @param ESP_OK - success, other - failed

esp_err_t esp_blufi_send_error_info(esp_blufi_error_state_t state)

/// 
/// This function is called to custom data.
///
/// @param ESP_OK - success, other - failed

esp_err_t esp_blufi_send_custom_data(uint8_t *data, uint32_t data_len)
```

---

Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151
Chapter 2. API

- **data** - custom data value
- **data_len** - the length of custom data

返回 ESP_OK - success, other - failed

Unions

union `esp_blufi_cb_param_t`

```
#include <esp_blufi_api.h> BLUFI callback parameters union.
```

Public Members

```c
struct esp_blufi_cb_param_t::blufi_init_finish_evt_param init_finish
    Blufi callback param of ESP_BLUFI_EVENT_INIT_FINISH

struct esp_blufi_cb_param_t::blufi_deinit_finish_evt_param deinit_finish
    Blufi callback param of ESP_BLUFI_EVENT_DEINIT_FINISH

struct esp_blufi_cb_param_t::blufi_set_wifi_mode_evt_param wifi_mode
    Blufi callback param of ESP_BLUFI_EVENT_INIT_FINISH

struct esp_blufi_cb_param_t::blufi_connect_evt_param connect
    Blufi callback param of ESP_BLUFI_EVENT_CONNECT

struct esp_blufi_cb_param_t::blufi_disconnect_evt_param disconnect
    Blufi callback param of ESP_BLUFI_EVENT_DISCONNECT

struct esp_blufi_cb_param_t::blufi_recv_sta_bssid_evt_param sta_bssid
    Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_BSSID

struct esp_blufi_cb_param_t::blufi_recv_sta_ssid_evt_param sta_ssid
    Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_SSID

struct esp_blufi_cb_param_t::blufi_recv_sta_passwd_evt_param sta_passwd
    Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_PASSWD

struct esp_blufi_cb_param_t::blufi_recv_softap_ssid_evt_param softap_ssid
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_SSID

struct esp_blufi_cb_param_t::blufi_recv_softap_passwd_evt_param softap_passwd
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD

struct esp_blufi_cb_param_t::blufi_recv_softap_max_conn_num_evt_param softap_max_conn_num
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM

struct esp_blufi_cb_param_t::blufi_recv_softap_auth_mode_evt_param softap_auth_mode
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE

struct esp_blufi_cb_param_t::blufi_recv_softap_channel_evt_param softap_channel
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL
```
struct \textit{esp\_blufi\_cb\_param\_t::blufi\_recv\_username\_evt\_param username}  
Blufi callback param of ESP\_BLUFI\_EVENT\_RECV\_USERNAME

struct \textit{esp\_blufi\_cb\_param\_t::blufi\_recv\_ca\_evt\_param ca}  
Blufi callback param of ESP\_BLUFI\_EVENT\_RECV\_CA\_CERT

struct \textit{esp\_blufi\_cb\_param\_t::blufi\_recv\_client\_cert\_evt\_param client\_cert}  
Blufi callback param of ESP\_BLUFI\_EVENT\_RECV\_CLIENT\_CERT

struct \textit{esp\_blufi\_cb\_param\_t::blufi\_recv\_server\_cert\_evt\_param server\_cert}  
Blufi callback param of ESP\_BLUFI\_EVENT\_RECV\_SERVER\_CERT

struct \textit{esp\_blufi\_cb\_param\_t::blufi\_recv\_client\_pkey\_evt\_param client\_pkey}  
Blufi callback param of ESP\_BLUFI\_EVENT\_RECV\_CLIENT\_PRIV\_KEY

struct \textit{esp\_blufi\_cb\_param\_t::blufi\_recv\_server\_pkey\_evt\_param server\_pkey}  
Blufi callback param of ESP\_BLUFI\_EVENT\_RECV\_SERVER\_PRIV\_KEY

struct \textit{esp\_blufi\_cb\_param\_t::blufi\_get\_error\_evt\_param report\_error}  
Blufi callback param of ESP\_BLUFI\_EVENT\_REPORT\_ERROR

struct \textit{esp\_blufi\_cb\_param\_t::blufi\_recv\_custom\_data\_evt\_param custom\_data}  
Blufi callback param of ESP\_BLUFI\_EVENT\_RECV\_CUSTOM\_DATA

struct \textit{blufi\_connect\_evt\_param}  
\#include \textit{<esp\_blufi\_api\_h>} ESP\_BLUFI\_EVENT\_CONNECT.

**Public Members**

\textit{esp\_blufi\_bd\_addr\_t remote\_bda}  
Blufi Remote bluetooth device address

uint8\_t \textit{server\_if}  
server interface

uint16\_t \textit{conn\_id}  
Connection id

struct \textit{blufi\_deinit\_finish\_evt\_param}  
\#include \textit{<esp\_blufi\_api\_h>} ESP\_BLUFI\_EVENT\_DEINIT\_FINISH.

**Public Members**

\textit{esp\_blufi\_deinit\_state\_t state}  
De-initial status

struct \textit{blufi\_disconnect\_evt\_param}  
\#include \textit{<esp\_blufi\_api\_h>} ESP\_BLUFI\_EVENT\_DISCONNECT.
Public Members

`esp_blufi_bd_addr_t remote_bda`
Blufi Remote bluetooth device address

```
#include <esp_blufi_api.h>
```

ESPBLLUIEVENT_REPORT_ERROR.

Public Members

`esp_blufi_error_state_t state`
Blufi error state

```
#include <esp_blufi_api.h>
```

ESPBLLUIEVENI_INIT_FINISH.

Public Members

`esp_blufi_init_state_t state`
Initial status

```
#include <esp_blufi_api.h>
```

ESPBLLUIEVENI_RECV_CA_CERT.

Public Members

`uint8_t *cert`
CA certificate point

`int cert_len`
CA certificate length

```
#include <esp_blufi_api.h>
```

ESPBLLUIEVENI_RECV_CLIENT_CERT

Public Members

`uint8_t *cert`
Client certificate point

`int cert_len`
Client certificate length

```
#include <esp_blufi_api.h>
```

ESPBLLUIEVENI_RECV_CLIENT_PRIV_KEY
Public Members

uint8_t *pkey
Client Private Key point, if Client certificate not contain Key

int pkey_len
Client Private key length

struct blufi_recv_custom_data_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CUSTOM_DATA.

Public Members

uint8_t *data
Custom data

uint32_t data_len
Custom data Length

struct blufi_recv_server_cert_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SERVER_CERT

Public Members

uint8_t *cert
Client certificate point

int cert_len
Client certificate length

struct blufi_recv_server_pkey_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY

Public Members

uint8_t *pkey
Client Private Key point, if Client certificate not contain Key

int pkey_len
Client Private key length

struct blufi_recv_softap_auth_mode_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE.
Public Members

\texttt{wifi\_auth\_mode\_t auth\_mode}
Authentication mode

\texttt{struct blufi\_recv\_softap\_channel\_evt\_param}
#include <esp\_blufi\_api\_h> ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_CHANNEL.

Public Members

\texttt{uint8\_t channel}
Authentication mode

\texttt{struct blufi\_recv\_softap\_max\_conn\_num\_evt\_param}
#include <esp\_blufi\_api\_h> ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_MAX\_CONN\_NUM.

Public Members

\texttt{int max\_conn\_num}
SSID

\texttt{struct blufi\_recv\_softap\_passwd\_evt\_param}
#include <esp\_blufi\_api\_h> ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_PASSWD.

Public Members

\texttt{uint8\_t \*passwd}
Password

\texttt{int passwd\_len}
Password Length

\texttt{struct blufi\_recv\_softap\_ssid\_evt\_param}
#include <esp\_blufi\_api\_h> ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_SSID.

Public Members

\texttt{uint8\_t \*ssid}
SSID

\texttt{int ssid\_len}
SSID length

\texttt{struct blufi\_recv\_sta\_bssid\_evt\_param}
#include <esp\_blufi\_api\_h> ESP\_BLUFI\_EVENT\_RECV\_STA\_BSSID.
Public Members

uint8_t *bssid[6]
BSSID

struct blufi_recv_sta_passwd_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_PASSWD.

Public Members

uint8_t *passwd
Password

int passwd_len
Password Length

struct blufi_recv_sta_ssid_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_SSID.

Public Members

uint8_t *ssid
SSID

int ssid_len
SSID length

struct blufi_recv_username_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_USERNAME.

Public Members

uint8_t *name
Username point

int name_len
Username length

struct blufi_set_wifi_mode_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_SET_WIFI_MODE.

Public Members

wifi_mode_t op_mode
Wifi operation mode
Chapter 2. API 参考

Structures

struct esp_blufi_extra_info_t
BLUFI extra information structure.

Public Members

uint8_t sta_bssid[6]
BSSID of station interface

bool sta_bssid_set
is BSSID of station interface set

uint8_t* sta_ssid
SSID of station interface

int sta_ssid_len
length of SSID of station interface

uint8_t* sta_passwd
password of station interface

int sta_passwd_len
length of password of station interface

uint8_t* softap_ssid
SSID of softap interface

int softap_ssid_len
length of SSID of softap interface

uint8_t* softap_passwd
password of station interface

int softap_passwd_len
length of password of station interface

uint8_t softap_authmode
authentication mode of softap interface

bool softap_authmode_set
is authentication mode of softap interface set

uint8_t softap_max_conn_num
max connection number of softap interface

bool softap_max_conn_num_set
is max connection number of softap interface set
uint8_t softap_channel
    channel of softap interface

bool softap_channel_set
    is channel of softap interface set

uint8_t sta_max_conn_retry
    max retry of sta establish connection

bool sta_max_conn_retry_set
    is max retry of sta establish connection set

uint8_t sta_conn_end_reason
    reason of sta connection end

bool sta_conn_end_reason_set
    is reason of sta connection end set

int8_t sta_conn_rssi
    rssiof sta connection

bool sta_conn_rssi_set
    is rssiof sta connection set

struct esp_blufi_ap_record_t
    Description of an WiFi AP.

Public Members

uint8_t ssid[33]
    SSID of AP

int8_t rssi
    signal strength of AP

struct esp_blufi_callbacks_t
    BLUFI callback functions type.

Public Members

esp_blufi_event_cb_t event_cb
    BLUFI event callback

esp_blufi_negotiate_data_handler_t negotiate_data_handler
    BLUFI negotiate data function for negotiate share key

esp_blufi_encrypt_func_t encrypt_func
    BLUFI encrypt data function with share key generated by negotiate_data_handler
**ESPIF**

### Chapter 2. API Reference

#### Macros

**ESP BLUFI BD ADDR LEN**

Bluetooth address length.

#### Type Definitions

**typedef uint8_t esp_blufi_bd_addr_t[ESP_BLUFI_BD_ADDR_LEN]**

Bluetooth device address.

**typedef void(*esp_blufi_event_cb_t)(esp_blufi_cb_event_t event, esp_blufi_cb_param_t *param)**

BLUFI event callback function type.

- **Param event**: Event type
- **Param param**: Point to callback parameter, currently is union type

**typedef void(*esp_blufi_negotiate_data_handler_t)(uint8_t* data, int len, uint8_t** output_data, int *output_len, bool *need_free)**

BLUFI negotiate data handler.

- **Param data**: Data from phone
- **Param len**: Length of data from phone
- **Param output_data**: Data want to send to phone
- **Param output_len**: Length of data want to send to phone
- **Param need_free**: Output reporting if memory needs to be freed or not

**typedef int(*esp_blufi_encrypt_func_t)(uint8_t iv8, uint8_t* crypt_data, int crypt_len)**

BLUFI encrypt the data after negotiate a share key.

- **Param iv8**: Initial vector(8bit), normally, blufi core will input packet sequence number
- **Param crypt_data**: Plain text and encrypted data, the encrypt function must support autothonous encrypt
- **Param crypt_len**: Length of plain text
- **Return**: Nonnegative number is encrypted length, if error, return negative number;

**typedef int(*esp_blufi_decrypt_func_t)(uint8_t iv8, uint8_t* crypt_data, int crypt_len)**

BLUFI decrypt the data after negotiate a share key.

- **Param iv8**: Initial vector(8bit), normally, blufi core will input packet sequence number
- **Param crypt_data**: Encrypted data and plain text, the encrypt function must support autothonous decrypt
- **Param crypt_len**: Length of encrypted text
- **Return**: Nonnegative number is decrypted length, if error, return negative number;

**typedef uint16_t(*esp_blufi_checksum_func_t)(uint8_t iv8, uint8_t* data, int len)**

BLUFI checksum.

- **Param iv8**: Initial vector(8bit), normally, blufi core will input packet sequence number
- **Param data**: Data need to checksum
- **Param len**: Length of data
Enumerations

e num esp_blufi_cb_event_t

Values:

enumerator ESP_BLUFI_EVENT_INIT_FINISH
enumerator ESP_BLUFI_EVENT_DEINIT_FINISH
enumerator ESP_BLUFI_EVENT_SET_WIFI_OPMODE
enumerator ESP_BLUFI_EVENT_BLE_CONNECT
enumerator ESP_BLUFI_EVENT_BLE_DISCONNECT
enumerator ESP_BLUFI_EVENT_REQ_CONNECT_TO_AP
enumerator ESP_BLUFI_EVENT_REQ_DISCONNECT_FROM_AP
enumerator ESP_BLUFI_EVENT_GET_WIFI_STATUS
enumerator ESP_BLUFI_EVENT_DEAUTHENTICATE_STA
enumerator ESP_BLUFI_EVENT_RECV_STA_BSSID
enumerator ESP_BLUFI_EVENT_RECV_STA_SSID
enumerator ESP_BLUFI_EVENT_RECV_STA_PASSWD
enumerator ESP_BLUFI_EVENT_RECV_SOFTAP_SSID
enumerator ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD
enumerator ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM
enumerator ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE
enumerator ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL
enumerator ESP_BLUFI_EVENT_RECV_USERNAME
enumerator ESP_BLUFI_EVENT_RECV_CA_CERT
enumerator ESP_BLUFI_EVENT_RECV_CLIENT_CERT
enumerator ESP_BLUFI_EVENT_RECV_SERVER_CERT
Chapter 2. API

enumerator ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY
enumerator ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY
enumerator ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE
enumerator ESP_BLUFI_EVENT_RECV_CUSTOM_DATA
enumerator ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE
enumerator ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY
enumerator ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY
enumerator ESP_BLUFI_EVENT_RECV_CUSTOM_DATA

enum esp_blufi_sta_conn_state_t
    BLUFI config status.
    Values:
    enumerator ESP_BLUFI_STA_CONN_SUCCESS
    enumerator ESP_BLUFI_STA_CONN_FAIL
    enumerator ESP_BLUFI_STA_CONNECTING
    enumerator ESP_BLUFI_STA_NO_IP

enum esp_blufi_init_state_t
    BLUFI init status.
    Values:
    enumerator ESP_BLUFI_INIT_OK
    enumerator ESP_BLUFI_INIT_FAILED

enum esp_blufi_deinit_state_t
    BLUFI deinit status.
    Values:
    enumerator ESP_BLUFI_DEINIT_OK
    enumerator ESP_BLUFI_DEINIT_FAILED

enum esp_blufi_error_state_t
    Values:
    enumerator ESP_BLUFI_SEQUENCE_ERROR
    enumerator ESP_BLUFI_CHECKSUM_ERROR
2.3.3 CLASSIC BT

CLASSIC BLUETOOTH GAP API

API Reference

Header File

- components/bt/host/bluedroid/api/include/api/esp_gap_bt_api.h

Functions

static inline uint32_t esp_bt_gap_get_cod_srvc(uint32_t cod)
get major service field of COD
参数 cod - [in] Class of Device
返回 major service bits

static inline uint32_t esp_bt_gap_get_cod_major_dev(uint32_t cod)
get major device field of COD
参数 cod - [in] Class of Device
返回 major device bits

static inline uint32_t esp_bt_gap_get_cod_minor_dev(uint32_t cod)
get minor service field of COD
参数 cod - [in] Class of Device
返回 minor service bits

static inline uint32_t esp_bt_gap_get_cod_format_type(uint32_t cod)
get format type of COD
参数 cod - [in] Class of Device
返回 format type
static inline bool esp_bt_gap_is_valid_cod(uint32_t cod)
    decide the integrity of COD
参数 cod[in] Class of Device
返回
• true if cod is valid
• false otherwise

esp_err_t esp_bt_gap_register_callback(esp_bt_gap_cb_t callback)
register callback function. This function should be called after esp_bluedroid_enable() completes successfully
返回
• ESP_OK: Succeed
• ESP_FAIL: others

esp_err_t esp_bt_gap_set_scan_mode(esp_bt_connection_mode_t c_mode, esp_bt_discovery_mode_t d_mode)
Set discoverability and connectability mode for legacy bluetooth. This function should be called after esp_bluedroid_enable() completes successfully.
参数
• c_mode[in]: one of the enums of esp_bt_connection_mode_t
• d_mode[in]: one of the enums of esp_bt_discovery_mode_t
返回
• ESP_OK: Succeed
• ESP_ERR_INVALID_ARG: if argument invalid
• ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others

esp_err_t esp_bt_gap_start_discovery(esp_bt_inq_mode_t mode, uint8_t inq_len, uint8_t num_rsps)
This function starts Inquiry and Name Discovery. This function should be called after esp_bluedroid_enable() completes successfully. When Inquiry is halted and cached results do not contain device name, then Name Discovery will connect to the peer target to get the device name. esp_bt_gap_cb_t will be called with ESP_BT_GAP_DISC_STATE_CHANGED_EVT when Inquiry is started or Name Discovery is completed. esp(bt_gap_cb_t will be called with ESP_BT_GAP_DISC_RES_EVT each time the two types of discovery results are got.
参数
• mode[in]: Inquiry mode
• inq_len[in]: Inquiry duration in 1.28 sec units, ranging from 0x01 to 0x30. This parameter only specifies the total duration of the Inquiry process, when this time expires, Inquiry will be halted.
• num_rsps[in]: Number of responses that can be received before the Inquiry is halted, value 0 indicates an unlimited number of responses.
返回
• ESP_OK: Succeed
• ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_ERR_INVALID_ARG: if invalid parameters are provided
• ESP_FAIL: others

esp_err_t esp_bt_gap_cancel_discovery()
Cancel Inquiry and Name Discovery. This function should be called after esp_bluedroid_enable() completes successfully. esp(bt_gap_cb_t will be called with ESP_BT_GAP_DISC_STATE_CHANGED_EVT if Inquiry or Name Discovery is cancelled by calling this function.
返回
• ESP_OK: Succeed
• ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others
**esp_err_t esp_bt_gap_get_remote_services**(esp_bd_addr_t remote_bda)

Start SDP to get remote services. This function should be called after esp_bluedroid_enable() completes successfully. esp_bt_gap_cb_t will be called with ESP_BT_GAP_RMT_SRVCS_EVT after service discovery ends.

- **ESP_OK**: Succeed
- **ESP_ERR_INVALID_STATE**: if bluetooth stack is not yet enabled
- **ESP_FAIL**: others

**esp_err_t esp_bt_gap_get_remote_service_record**(esp_bd_addr_t remote_bda, esp_bt_uuid_t *uuid)

Start SDP to look up the service matching uuid on the remote device. This function should be called after esp_bluedroid_enable() completes successfully.

esp_bt_gap_cb_t will be called with ESP_BT_GAP_RMT_SRVC_REC_EVT after service discovery ends.

- **ESP_OK**: Succeed
- **ESP_ERR_INVALID_STATE**: if bluetooth stack is not yet enabled
- **ESP_FAIL**: others

**esp_err_t esp_bt_gap_resolve_eir_data**(uint8_t*eir, esp_bt_eir_type_t type, uint8_t*length)

This function is called to get EIR data for a specific type.

- **eir** - [in] - pointer of raw eir data to be resolved
- **type** - [in] - specific EIR data type
- **length** - [out] - return the length of EIR data excluding fields of length and data type

- **ESP_OK**: Succeed
- **ESP_ERR_INVALID_STATE**: if bluetooth stack is not yet enabled
- **ESP_ERR_INVALID_ARG**: if param is invalid
- **ESP_FAIL**: others

**esp_err_t esp_bt_gap_config_eir_data**(esp_bt_eir_data_t *eir_data)

This function is called to config EIR data.

- **eir_data** - [in] - pointer of EIR data content

- **ESP_OK**: Succeed
- **ESP_ERR_INVALID_STATE**: if bluetooth stack is not yet enabled
- **ESP_ERR_INVALID_ARG**: if param is invalid
- **ESP_FAIL**: others

**esp_err_t esp_bt_gap_set_cod**(esp_bt_cod_t cod, esp_bt_cod_mode_t mode)

This function is called to set class of device. The structure esp_bt_gapCb_t will be called with ESP_BT_GAP_SET_COD_EVT after set COD ends. Some profile have special restrictions on class of device, changes may cause these profile do not work.

- **cod** - [in] - class of device
- **mode** - [in] - setting mode

- **ESP_OK**: Succeed
- **ESP_ERR_INVALID_STATE**: if bluetooth stack is not yet enabled
- **ESP_ERR_INVALID_ARG**: if param is invalid
- **ESP_FAIL**: others
```c
esp_err_t esp_bt_gap_get_cod(esp_bt_cod_t *cod)
This function is called to get class of device.
参数 cod - [out] - class of device
返回
• ESP_OK : Succeed
• ESP_FAIL : others
```

```c
esp_err_t esp_bt_gap_read_rssi_delta(esp_bd_addr_t remote_addr)
This function is called to read RSSI delta by address after connected. The RSSI value returned by ESP_BT_GAP_READ_RSSI_DELTA_EVT.
参数 remote_addr - [in] - remote device address, corresponding to a certain connection handle
返回
• ESP_OK : Succeed
• ESP_FAIL : others
```

```c
esp_err_t esp_bt_gap_remove_bond_device(esp_bd_addr_t bd_addr)
Removes a device from the security database list of peer device.
参数 bd_addr - [in] : BD address of the peer device
返回 - ESP_OK : success
• ESP_FAIL : failed
```

```c
int esp_bt_gap_get_bond_device_num(
Get the device number from the security database list of peer device. It will return the device bonded number immediately.
返回 - >= 0 : bonded devices number
• ESP_FAIL : failed
```

```c
esp_err_t esp_bt_gap_get_bond_device_list(int *dev_num, esp_bd_addr_t *dev_list)
Get the device from the security database list of peer device. It will return the device bonded information immediately.
参数
• dev_num - [inout] Indicate the dev_list array(buffer) size as input. If dev_num is large enough, it means the actual number as output. Suggest that dev_num value equal to esp_ble_get_bond_device_num().
• dev_list - [out] an array(buffer) of esp_bd_addr_t type. Use for storing the bonded devices address. The dev_list should be allocated by who call this API.
返回
• ESP_OK : Succeed
• ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others
```

```c
esp_err_t esp_bt_gap_set_pin(esp_bt_pin_type_t pin_type, uint8_t pin_code_len, esp_bt_pin_code_t pin_code)
Set pin type and default pin code for legacy pairing.
参数
• pin_type - [in] Use variable or fixed pin. If pin_type is ESP_BT_PIN_TYPE_VARIABLE, pin_code and pin_code_len will be ignored, and ESP_BT_GAP_PIN_REQ_EVT will come when control requests for pin code. Else, will use fixed pin code and not callback to users.
• pin_code_len - [in] Length of pin_code
• pin_code - [in] Pin_code
返回 - ESP_OK : success
• ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
• other : failed
```
esp_err_t esp_bt_gap_pin_reply (esp_bd_addr_t bd_addr, bool accept, uint8_t pin_code_len, esp_bt_pin_code_t pin_code)

Reply the pin_code to the peer device for legacy pairing when ESP_BT_GAP_PIN_REQ_EVT is coming.

参数
- bd_addr [in] BD address of the peer
- accept [in] Pin_code reply successful or declined.
- pin_code_len [in] Length of pin_code
- pin_code [in] Pin_code

返回
- ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other: failed

esp_err_t esp_bt_gap_set_security_param (esp_bt_sp_param_t param_type, void* value, uint8_t len)

Set a GAP security parameter value. Overrides the default value.

参数
- param_type [in]: the type of the param which is to be set
- value [in]: the param value
- len [in]: the length of the param value

返回
- ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other: failed

esp_err_t esp_bt_gap_ssp_passkey_reply (esp_bd_addr_t bd_addr, bool accept, uint32_t passkey)

Reply the key value to the peer device in the legacy connection stage.

参数
- bd_addr [in]: BD address of the peer
- accept [in]: passkey entry successful or declined.
- passkey [in]: passkey value, must be a 6 digit number, can be lead by 0.

返回
- ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other: failed

esp_err_t esp_bt_gap_ssp_confirm_reply (esp_bd_addr_t bd_addr, bool accept)

Reply the confirm value to the peer device in the legacy connection stage.

参数
- bd_addr [in]: BD address of the peer device
- accept [in]: numbers to compare are the same or different

返回
- ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other: failed

esp_err_t esp_bt_gap_set_afh_channels (esp_bt_gap_afh_channels channels)

Set the AFH channels.

参数 channels [in]: The n th such field (in the range 0 to 78) contains the value for channel n
: 0 means channel n is bad. 1 means channel n is unknown. The most significant bit is reserved
and shall be set to 0. At least 20 channels shall be marked as unknown.

返回
- ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other: failed

esp_err_t esp_bt_gap_read_remote_name (esp_bd_addr_t remote_bda)

Read the remote device name.

参数 remote_bda [in] The remote device’s address

返回
- ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other: failed
esp_err_t esp_bt_gap_set_qos (esp_bd_addr_t remote_bda, uint32_t t_poll)

Config Quality of service.

参数
- remote_bda [in] The remote device’s address
- t_poll [in] Poll interval, the maximum time between transmissions which from the master to a particular slave on the ACL logical transport. unit is 0.625ms

返回
- ESP_OK : success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Unions

union esp_bt_gap_cb_param_t

#include <esp_gap_bt_api.h> GAP state callback parameters.

Public Members

struct esp_bt_gap_cb_param_t::disc_res_param disc_res
discovery result parameter struct

struct esp_bt_gap_cb_param_t::disc_state_changed_param disc_st_chg
discovery state changed parameter struct

struct esp_bt_gap_cb_param_t::rmt_srvcs_param rmt_srvcs
services of remote device parameter struct

struct esp_bt_gap_cb_param_t::rmt_svc_rec_param rmt_svc_rec
specific service record from remote device parameter struct

struct esp_bt_gap_cb_param_t::read_rssi_delta_param read_rssi_delta
read rssi parameter struct

struct esp_bt_gap_cb_param_t::config_eir_data_param config_eir_data
config EIR data

struct esp_bt_gap_cb_param_t::auth_cmpl_param auth_cmpl
authentication complete parameter struct

struct esp_bt_gap_cb_param_t::pin_req_param pin_req
pin request parameter struct

struct esp_bt_gap_cb_param_t::cfm_req_param cfm_req
confirm request parameter struct

struct esp_bt_gap_cb_param_t::keynotif_param keynotif
passkey notif parameter struct

struct esp_bt_gap_cb_param_t::key_req_param key_req
passkey request parameter struct
struct esp_bt_gap_cb_param_t::set_afh_channels_param set_afh_channels
   set AFH channel parameter struct

struct esp_bt_gap_cb_param_t::read_rmt_name_param read_rmt_name
   read Remote Name parameter struct

struct esp_bt_gap_cb_param_t::mode_chg_param mode_chg
   mode change event parameter struct

struct esp_bt_gap_cb_param_t::bt_remove_bond_dev_cmpl_evt_param remove_bond_dev_cmpl
   Event parameter of ESP_BT_GAP_REMOVE_BOND_DEV_COMPLETE_EVT

struct esp_bt_gap_cb_param_t::qos_cmpl_param qos_cmpl
   QoS complete parameter struct

struct auth_cmpl_param
   #include <esp_gap_bt_api.h> ESP_BT_GAP_AUTH_CMPL_EVT.

Public Members

   esp_bd_addr_t bda
      remote bluetooth device address

   esp_bt_status_t stat
      authentication complete status

   uint8_t device_name[ESP_BT_GAP_MAX_BDNAME_LEN + 1]
      device name

struct bt_remove_bond_dev_cmpl_evt_param
   #include <esp_gap_bt_api.h> ESP_BT_GAP_REMOVE_BOND_DEV_COMPLETE_EVT.

Public Members

   esp_bd_addr_t bda
      remote bluetooth device address

   esp_bt_status_t status
      Indicate the remove bond device operation success status

struct cfm_req_param
   #include <esp_gap_bt_api.h> ESP_BT_GAP_CFM_REQ_EVT.

Public Members
```c
esp_bd_addr_t bda
    remote bluetooth device address

uint32_t num_val
    the numeric value for comparison.
```

```c
struct config_eir_data_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_CONFIG_EIR_DATA_EVT *
```

### Public Members

```c
esp_bt_status_t stat
    config EIR status: ESP_BT_STATUS_SUCCESS: config success
    ESP_BT_STATUS_EIR_TOO_LARGE: the EIR data is more than 240B. The EIR may not
    contain the whole data. others: failed
```

```c
uint8_t eir_type_num
    the number of EIR types in EIR type
```

```c
esp_bt_eir_type_t eir_type[ESP_BT_EIR_TYPE_MAX_NUM]
    EIR types in EIR type
```

```
struct disc_res_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_DISC_RES_EVT.
```

### Public Members

```c
esp_bd_addr_t bda
    remote bluetooth device address
```

```c
int num_prop
    number of properties got
```

```c
esp_bt_gap_dev_prop_t *prop
    properties discovered from the new device
```

```
struct disc_state_changed_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_DISC_STATE_CHANGED_EVT.
```

### Public Members

```c
esp_bt_gap_discovery_state_t state
    discovery state
```

```
struct key_notif_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_KEY_NOTIF_EVT.
```
Public Members

`esp_bd_addr_t bda`
remote bluetooth device address

`uint32_t passkey`
the numeric value for passkey entry.

```c
struct key_req_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_KEY_REQ_EVT.
```

Public Members

`esp_bd_addr_t bda`
remote bluetooth device address

```c
struct mode_chg_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_MODE_CHG_EVT.
```

Public Members

`esp_bd_addr_t bda`
remote bluetooth device address

```c
struct pin_req_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_PIN_REQ_EVT.
```

Public Members

`esp_bd_addr_t bda`
remote bluetooth device address

```c
bool min_16_digit
TRUE if the pin returned must be at least 16 digits
```

```c
struct qos_cmpl_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_QOS_CMPL_EVT.
```

Public Members

`esp_bt_status_t stat`
QoS status
\texttt{esp\_bd\_addr\_t \ bda}

remote bluetooth device address

\texttt{uint32\_t \ t\_poll}

poll interval, the maximum time between transmissions which from the master to a particular slave on the ACL logical transport. unit is 0.625ms.

\textbf{Public Members}

\texttt{esp\_bd\_addr\_t \ bda}

remote bluetooth device address

\texttt{esp\_bt\_status\_t \ stat}

read Remote Name status

\texttt{uint8\_t \ rmt\_name[ESP\_BT\_GAP\_MAX\_BDNAME\_LEN + 1]}

Remote device name

\textbf{struct \texttt{read\_rmt\_name\_param}}

\texttt{#include \textless esp\_gap\_bt\_api.h\textgreater} ESP\_BT\_GAP\_READ\_REMOTE\_NAME\_EVT.

\textbf{Public Members}

\texttt{esp\_bd\_addr\_t \ bda}

remote bluetooth device address

\texttt{esp\_bt\_status\_t \ stat}

read rssi status

\texttt{int8\_t \ rssi\_delta}

rssi delta value range -128 ~ 127. The value zero indicates that the RSSI is inside the Golden Receive Power Range, the Golden Receive Power Range is from ESP\_BT\_GAP\_RSSI\_LOW\_THRLD to ESP\_BT\_GAP\_RSSI\_HIGH\_THRLD

\textbf{struct \texttt{rmt\_srvc\_rec\_param}}

\texttt{#include \textless esp\_gap\_bt\_api.h\textgreater} ESP\_BT\_GAP\_RMT\_SRVC\_REC\_EVT.

\textbf{Public Members}

\texttt{esp\_bd\_addr\_t \ bda}

remote bluetooth device address

\texttt{esp\_bt\_status\_t \ stat}

service search status


```c
struct rmt_srvcs_param
    #include <esp_gap_bt_api.h> ESP_BT_GAP_RMT_SRVCS_EVT.

Public Members

esp_bd_addr_t bda
    remote bluetooth device address

esp_bt_status_t stat
    service search status

int num_uuids
    number of UUID in uuid_list

esp_bt_uuid_t *uuid_list
    list of service UUIDs of remote device

struct set_afh_channels_param
    #include <esp_gap_bt_api.h> ESP_BT_GAP_SET_AFHCHANNELS_EVT.

Public Members

esp_bt_status_t stat
    set AFH channel status

Structures
struct esp_bt_cod_t
    Class of device.

Public Members

uint32_t reserved_2
    undefined

uint32_t minor
    minor class

uint32_t major
    major class

uint32_t service
    service class

uint32_t reserved_8
    undefined
```
struct esp_bt_gap_dev_prop_t
   Bluetooth Device Property Descriptor.

Public Members

   esp_bt_gap_dev_prop_type_t type
      Device property type

   int len
      Device property value length

   void *val
      Device property value

struct esp_bt_eir_data_t
   EIR data content, according to “Supplement to the Bluetooth Core Specification”.

Public Members

   bool fec_required
      FEC is required or not, true by default

   bool include_txpower
      EIR data include TX power, false by default

   bool include_uuid
      EIR data include UUID, false by default

   uint8_t flag
      EIR flags, see ESP_BT_EIR_FLAG for details, EIR will not include flag if it is 0, 0 by default

   uint16_t manufacturer_len
      Manufacturer data length, 0 by default

   uint8_t *p_manufacturer_data
      Manufacturer data point

   uint16_t url_len
      URL length, 0 by default

   uint8_t *p_url
      URL point

Macros

ESP_BT_GAP_RSSI_HIGH_THRLD
   RSSI threshold.
   High RSSI threshold
Chapter 2. API

ESP_BT_GAP_RSSI_LOW_THRLD
Low RSSI threshold

ESP_BT_GAP_AFH_CHANNELS_LEN

ESP_BT_GAP_MAX_BDNAME_LEN
Maximum bytes of Bluetooth device name.

ESP_BT_GAP_EIR_DATA_LEN
Maximum size of EIR Significant part.

ESP_BT_EIR_TYPE_FLAGS
Extended Inquiry Response data type.
Flag with information such as BR/EDR and LE support

ESP_BT_EIR_TYPE_INCMPL_16BITS_UUID
Incomplete list of 16-bit service UUIDs

ESP_BT_EIR_TYPE_CMPL_16BITS_UUID
Complete list of 16-bit service UUIDs

ESP_BT_EIR_TYPE_INCMPL_32BITS_UUID
Incomplete list of 32-bit service UUIDs

ESP_BT_EIR_TYPE_CMPL_32BITS_UUID
Complete list of 32-bit service UUIDs

ESP_BT_EIR_TYPE_INCMPL_128BITS_UUID
Incomplete list of 128-bit service UUIDs

ESP_BT_EIR_TYPE_CMPL_128BITS_UUID
Complete list of 128-bit service UUIDs

ESP_BT_EIR_TYPE_SHORT_LOCAL_NAME
Shortened Local Name

ESP_BT_EIR_TYPE_CMPL_LOCAL_NAME
Complete Local Name

ESP_BT_EIR_TYPE_TX_POWER_LEVEL
Tx power level, value is 1 octet ranging from -127 to 127, unit is dBm

ESP_BT_EIR_TYPE_URL
Uniform resource identifier

ESP_BT_EIR_TYPE_MANU_SPECIFIC
Manufacturer specific data
ESP_BT_EIR_TYPE_MAX_NUM
    MAX number of EIR type

ESP_BT_EIR_FLAG_LIMIT_DISC

ESP_BT_EIR_FLAG_GEN_DISC

ESP_BT_EIR_FLAG_BREDR_NOT_SPT

ESP_BT_EIR_FLAG_DMT_CONTROLLER_SPT

ESP_BT_EIR_FLAG_DMT_HOST_SPT

ESP_BT_EIR_MAX_LEN

ESP_BT_PIN_CODE_LEN
    Max pin code length

ESP_BT_IO_CAP_OUT

ESP_BT_IO_CAP_IO

ESP_BT_IO_CAP_IN

ESP_BT_IO_CAP_NONE

ESP_BT_PM_MD_ACTIVE
    Active mode

ESP_BT_PM_MD_HOLD
    Hold mode

ESP_BT_PM_MD_SNIFF
    Sniff mode

ESP_BT_PM_MD_PARK
    Park state

ESP_BT_COD_SRVC_BIT_MASK
    Bits of major service class field.
    Major service bit mask

ESP_BT_COD_SRVC_BIT_OFFSET
    Major service bit offset
**ESP_BT_COD_MAJOR_DEV_BIT_MASK**
Bits of major device class field.
Major device bit mask

**ESP_BT_COD_MAJOR_DEV_BIT_OFFSET**
Major device bit offset

**ESP_BT_COD_MINOR_DEV_BIT_MASK**
Bits of minor device class field.
Minor device bit mask

**ESP_BT_COD_MINOR_DEV_BIT_OFFSET**
Minor device bit offset

**ESP_BT_COD_FORMAT_TYPE_BIT_MASK**
Bits of format type.
Format type bit mask

**ESP_BT_COD_FORMAT_TYPE_BIT_OFFSET**
Format type bit offset

**ESP_BT_COD_FORMAT_TYPE_1**
Class of device format type 1.

**ESP_BT_GAP_MIN_INQ_LEN**
Minimum and maximum inquiry length. Minimum inquiry duration, unit is 1.28s

**ESP_BT_GAP_MAX_INQ_LEN**
Maximum inquiry duration, unit is 1.28s

**Type Definitions**

typedef uint8_t esp_bt_gap_afh_channels[ESP_BT_GAP_AFH_CHANNELS_LEN]
typedef uint8_t esp_bt_eir_type_t

typedef uint8_t esp_bt_pin_code_t[ESP_BT_PIN_CODE_LEN]
Pin Code (upto 128 bits) MSB is 0

typedef uint8_t esp_bt_io_cap_t
Combination of the IO Capability

typedef uint8_t esp_bt_pm_mode_t

typedef void (*esp_bt_gap_cb_t)(esp_bt_gap_cb_event_t event, esp_bt_gap_cb_param_t *param)
bluetooth GAP callback function type

Param event : Event type
Param param : Pointer to callback parameter
Enumerations

enum `esp_bt_cod_mode_t`

class of device settings

Values:

enumerator `ESP_BT_SET_COD_MAJOR_MINOR`
overwrite major, minor class

enumerator `ESP_BT_SET_COD_SERVICE_CLASS`
set the bits in the input, the current bit will remain

enumerator `ESP_BT_CLR_COD_SERVICE_CLASS`
clear the bits in the input, others will remain

enumerator `ESP_BT_SET_COD_ALL`
overwrite major, minor, set the bits in service class

enumerator `ESP_BT_INIT_COD`
overwrite major, minor, and service class

enum `esp_bt_connection_mode_t`
Discoverability and Connectability mode.

Values:

enumerator `ESP_BT_NON_CONNECTABLE`
Non-connectable

enumerator `ESP_BT_CONNECTABLE`
Connectable

enum `esp_bt_discovery_mode_t`

Values:

enumerator `ESP_BT_NON_DISCOVERABLE`
Non-discoverable

enumerator `ESP_BT_LIMITED_DISCOVERABLE`
Limited Discoverable

enumerator `ESP_BT_GENERAL_DISCOVERABLE`
General Discoverable

enum `esp_bt_gap_dev_prop_type_t`
Bluetooth Device Property type.

Values:

enumerator `ESP_BT_GAP_DEV_PROP_BDNAME`
Bluetooth device name, value type is `int8_t []`
enumerator ESP_BT_GAP_DEV_PROP_COD
   Class of Device, value type is uint32_t

enumerator ESP_BT_GAP_DEV_PROP_RSSI
   Received Signal strength Indication, value type is int8_t, ranging from -128 to 127

enumerator ESP_BT_GAP_DEV_PROP_EIR
   Extended Inquiry Response, value type is uint8_t[]

enum esp_bt_cod_srvc_t
   Major service class field of Class of Device, multiple bits can be set.
   Values:

   enumerator ESP_BT_COD_SRVC_NONE
      None indicates an invalid value

   enumerator ESP_BT_COD_SRVC_LMTD_DISCOVER
      Limited Discoverable Mode

   enumerator ESP_BT_COD_SRVC_POSITIONING
      Positioning (Location identification)

   enumerator ESP_BT_COD_SRVC_NETWORKING
      Networking, e.g. LAN, Ad hoc

   enumerator ESP_BT_COD_SRVC_RENDERING
      Rendering, e.g. Printing, Speakers

   enumerator ESP_BT_COD_SRVC_CAPTURING
      Capturing, e.g. Scanner, Microphone

   enumerator ESP_BT_COD_SRVC_OBJ_TRANSFER
      Object Transfer, e.g. v-Inbox, v-Folder

   enumerator ESP_BT_COD_SRVC_AUDIO
      Audio, e.g. Speaker, Microphone, Headset service

   enumerator ESP_BT_COD_SRVC_TELEPHONY
      Telephony, e.g. Cordless telephony, Modem, Headset service

   enumerator ESP_BT_COD_SRVC_INFORMATION
      Information, e.g., WEB-server, WAP-server

enum esp_bt_pin_type_t
   Values:

   enumerator ESP_BT_PIN_TYPE_VARIABLE
      Refer to BTM_PIN_TYPE_VARIABLE
enumerator **ESP_BT_PIN_TYPE_FIXED**
   Refer to BTM_PIN_TYPE_FIXED

enum **esp_bt_sp_param_t**
   *Values:*

   enumerator **ESP_BT_SP_IOCAP_MODE**
      Set IO mode

enum **esp_bt_cod_major_dev_t**
   Major device class field of Class of Device.
   *Values:*

   enumerator **ESP_BT_COD_MAJOR_DEV_MISC**
      Miscellaneous

   enumerator **ESP_BT_COD_MAJOR_DEV_COMPUTER**
      Computer

   enumerator **ESP_BT_COD_MAJOR_DEV_PHONE**
      Phone (cellular, cordless, pay phone, modem

   enumerator **ESP_BT_COD_MAJOR_DEV_LAN_NAP**
      LAN, Network Access Point

   enumerator **ESP_BT_COD_MAJOR_DEV_AV**
      Audio/Video (headset, speaker, stereo, video display, VCR

   enumerator **ESP_BT_COD_MAJOR_DEV_PERIPHERAL**
      Peripheral (mouse, joystick, keyboard)

   enumerator **ESP_BT_COD_MAJOR_DEV_IMAGING**
      Imaging (printer, scanner, camera, display

   enumerator **ESP_BT_COD_MAJOR_DEV_WEARABLE**
      Wearable

   enumerator **ESP_BT_COD_MAJOR_DEV_TOY**
      Toy

   enumerator **ESP_BT_COD_MAJOR_DEV_HEALTH**
      Health

   enumerator **ESP_BT_COD_MAJOR_DEV_UNCATEGORIZED**
      Uncategorized: device not specified

enum **esp_bt_gap_discovery_state_t**
   Bluetooth Device Discovery state
   *Values:
enumerator **ESP_BT_GAP_DISCOVERY_STOPPED**
Device discovery stopped

enumerator **ESP_BT_GAP_DISCOVERY_STARTED**
Device discovery started

enum **esp_bt_gap_cb_event_t**
BT GAP callback events.

Values:

enumerator **ESP_BT_GAP_DISC_RES_EVT**
Device discovery result event

enumerator **ESP_BT_GAP_DISC_STATE_CHANGED_EVT**
Discovery state changed event

enumerator **ESP_BT_GAP_RMT_SRVCS_EVT**
Get remote services event

enumerator **ESP_BT_GAP_RMT_SRVC_REC_EVT**
Get remote service record event

enumerator **ESP_BT_GAP_AUTH_CMPL_EVT**
Authentication complete event

enumerator **ESP_BT_GAP_PIN_REQ_EVT**
Legacy Pairing Pin code request

enumerator **ESP_BT_GAP_CFM_REQ_EVT**
Security Simple Pairing User Confirmation request.

enumerator **ESP_BT_GAP_KEY_NOTIF_EVT**
Security Simple Pairing Passkey Notification

enumerator **ESP_BT_GAP_KEY_REQ_EVT**
Security Simple Pairing Passkey request

enumerator **ESP_BT_GAP_READ_RSSI_DELTA_EVT**
Read rssi event

enumerator **ESP_BT_GAP_CONFIG_EIR_DATA_EVT**
Config EIR data event

enumerator **ESP_BT_GAP_SET_AFH_CHANNELS_EVT**
Set AFH channels event

enumerator **ESP_BT_GAP_READ_REMOTE_NAME_EVT**
Read Remote Name event
enumerator ESP_BT_GAP_MODE_CHG_EVT

enumerator ESP_BT_GAP_REMOVE_BOND_DEV_COMPLETE_EVT
    remove bond device complete event

enumerator ESP_BT_GAP_QOS_CMPL_EVT
    QOS complete event

enumerator ESP_BT_GAP_EVT_MAX

eum esp_bt_inq_mode_t
    Inquiry Mode
    Values:

enumerator ESP_BT_INQ_MODE_GENERAL_INQUIRY
    General inquiry mode

enumerator ESP_BT_INQ_MODE_LIMITED_INQUIRY
    Limited inquiry mode

Bluetooth A2DP API

Application Example Check bluetooth/bluedroid/classic_bt folder in ESP-IDF examples, which contains the following application:

• This is a A2DP sink client demo. This demo can be discovered and connected by A2DP source device and receive the audio stream from remote device - bluetooth/bluedroid/classic_bt/a2dp_sink

API Reference

Header File

• components/bt/host/bluedroid/api/include/api/esp_a2dp_api.h

Functions

esp_err_t esp_a2d_register_callback (esp_a2d_cb_t callback)

Register application callback function to A2DP module. This function should be called only after esp_bluedroid_enable() completes successfully, used by both A2DP source and sink.

参数 callback [in] A2DP event callback function
返回
• ESP_OK: success
• ESP_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: if callback is a NULL function pointer

esp_err_t esp_a2d_sink_register_data_callback (esp_a2d_sink_data_cb_t callback)

Register A2DP sink data output function; For now the output is PCM data stream decoded from SBC format. This function should be called only after esp_bluedroid_enable() completes successfully, used only by A2DP sink. The callback is invoked in the context of A2DP sink task whose stack size is configurable through menuconfig.

参数 callback [in] A2DP sink data callback function
返回
Chapter 2. API

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

```
esp_err_t esp_a2d_sink_init (void)
```

Initialize the bluetooth A2DP sink module. This function should be called after esp_bluedroid_enable() completes successfully, and ESP_A2D_PROF_STATE_EVT with ESP_A2D_INIT_SUCCESS will reported to the APP layer. Note: A2DP can work independently. If you want to use AVRC together, you should initiate AVRC first. This function should be called after esp_bluedroid_enable() completes successfully.

返回
- ESP_OK: if the initialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```
esp_err_t esp_a2d_sink_deinit (void)
```

De-initialize for A2DP sink module. This function should be called only after esp_bluedroid_enable() completes successfully, and ESP_A2D_PROF_STATE_EVT with ESP_A2D_DEINIT_SUCCESS will reported to APP layer.

返回
- ESP_OK: if the deinitialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```
esp_err_t esp_a2d_sink_connect (esp_bd_addr_t remote_bda)
```

Connect to remote bluetooth A2DP source device. This API must be called after esp_a2d_sink_init() and before esp_a2d_sink_deinit().

参数 `remote_bda` [in] remote bluetooth device address

返回
- ESP_OK: connect request is sent to lower layer successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```
esp_err_t esp_a2d_sink_disconnect (esp_bd_addr_t remote_bda)
```

Disconnect from the remote A2DP source device. This API must be called after esp_a2d_sink_init() and before esp_a2d_sink_deinit().

参数 `remote_bda` [in] remote bluetooth device address

返回
- ESP_OK: disconnect request is sent to lower layer successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```
esp_err_t esp_a2d_sink_set_delay_value (uint16_t delay_value)
```

Set delay reporting value. The delay value of sink is caused by buffering (including protocol stack and application layer), decoding and rendering. The default delay value is 120ms, if the set value is less than 120ms, the setting will fail. This API must be called after esp_a2d_sink_init() and before esp_a2d_sink_deinit().

参数 `delay_value` [in] reporting value is in 1/10 millisecond

返回
- ESP_OK: delay value is sent to lower layer successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```
esp_err_t esp_a2d_sink_get_delay_value (void)
```

Get delay reporting value. This API must be called after esp_a2d_sink_init() and before esp_a2d_sink_deinit().

返回
- ESP_OK: if the request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others
esp_err_t esp_a2d_media_ctrl(esp_a2d_media_ctrl_t ctrl)

Media control commands. This API can be used for both A2DP sink and source and must be called after esp_a2d_sink_init() and before esp_a2d_sink_deinit().

参数 ctrl - [in] control commands for A2DP data channel

返回 • ESP_OK: control command is sent to lower layer successfully
• ESP_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others

esp_err_t esp_a2d_source_init(void)

Initialize the bluetooth A2DP source module. A2DP can work independently. If you want to use AVRC together, you should initiate AVRC first. This function should be called after esp_bluedroid_enable() completes successfully, and ESP_A2D_PROF_STATE_EVT with ESP_A2D_INIT_SUCCESS will reported to the APP layer. Note: A2DP can work independently. If you want to use AVRC together, you should initiate AVRC first.

返回 • ESP_OK: if the initialization request is sent to lower layer successfully
• ESP_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others

esp_err_t esp_a2d_source_deinit(void)

De-initialize for A2DP source module. This function should be called only after esp_bluedroid_enable() completes successfully, and ESP_A2D_PROF_STATE_EVT with ESP_A2D_DEINIT_SUCCESS will reported to APP layer.

返回 • ESP_OK: success
• ESP_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others

esp_err_t esp_a2d_source_register_data_callback(esp_a2d_source_data_cb_t callback)

Register A2DP source data input function. For now, the input should be PCM data stream. This function should be called only after esp_bluedroid_enable() completes successfully. The callback is invoked in the context of A2DP source task whose stack size is configurable through menuconfig.

参数 callback - [in] A2DP source data callback function

返回 • ESP_OK: success
• ESP_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: if callback is a NULL function pointer

esp_err_t esp_a2d_source_connect(esp_bd_addr_t remote_bda)

Connect to remote A2DP sink device. This API must be called after esp_a2d_source_init() and before esp_a2d_source_deinit().

参数 remote_bda - [in] remote bluetooth device address

返回 • ESP_OK: connect request is sent to lower layer successfully
• ESP_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others

esp_err_t esp_a2d_source_disconnect(esp_bd_addr_t remote_bda)

Disconnect from the remote A2DP sink device. This API must be called after esp_a2d_source_init() and before esp_a2d_source_deinit().

参数 remote_bda - [in] remote bluetooth device address

返回 • ESP_OK: disconnect request is sent to lower layer
• ESP_INVALID_STATE: if bluetooth stack is not yet enabled
• ESP_FAIL: others
Unions

union esp_a2d_cb_param_t
    #include <esp_a2dp_api.h> A2DP state callback parameters.

Public Members

struct esp_a2d_cb_param_t:a2d_conn_stat_param conn_stat
    A2DP connection status

struct esp_a2d_cb_param_t:a2d_audio_stat_param audio_stat
    audio stream playing state

struct esp_a2d_cb_param_t:a2d_audio_cfg_param audio_cfg
    media codec configuration information

struct esp_a2d_cb_param_t:media_ctrl_stat_param media_ctrl_stat
    status in acknowledgement to media control commands

struct esp_a2d_cb_param_t:a2d_prof_stat_param a2d_prof_stat
    status to indicate a2d prof init or deinit

struct esp_a2d_cb_param_t:a2d_psc_cfg_param a2d_psc_cfg_stat
    status to indicate protocol service capabilities configured

struct esp_a2d_cb_param_t:a2d_set_delay_value_param a2d_set_delay_value_stat
    A2DP sink set delay report value status

struct esp_a2d_cb_param_t:a2d_get_delay_param a2d_get_delay_value_stat
    A2DP sink get delay report value status

struct esp_a2d_cb_param_t:a2d_report_delay_stat_param a2d_report_delay_value_stat
    A2DP source received sink report value status

struct a2d_audio_cfg_param
    #include <esp_a2dp_api.h> ESP_A2D_AUDIO_CFG_EVT.

Public Members

esp_bd_addr_t remote_bda
    remote bluetooth device address

esp_a2d_mcc_t mcc
    A2DP media codec capability information

struct a2d_audio_stat_param
    #include <esp_a2dp_api.h> ESP_A2D_AUDIO_STATE_EVT.
Public Members

`esp_a2d_audio_state_t state`
   one of the values from esp_a2d_audio_state_t

`esp_bd_addr_t remote_bda`
   remote bluetooth device address

struct `a2d_conn_stat_param`
   #include `<esp_a2dp_api.h>` ESP_A2D_CONNECTION_STATE_EVT.

Public Members

`esp_a2d_connection_state_t state`
   one of values from esp_a2d_connection_state_t

`esp_bd_addr_t remote_bda`
   remote bluetooth device address

`esp_a2d_disc_rsn_t disc_rsn`
   reason of disconnection for “DISCONNECTED”

struct `a2d_get_stat_param`
   #include `<esp_a2dp_api.h>` ESP_A2D_SNK_GET_DELAY_VALUE_EVT.

Public Members

`uint16_t delay_value`
   delay report value

struct `a2d_prof_stat_param`
   #include `<esp_a2dp_api.h>` ESP_A2D_PROF_STATE_EVT.

Public Members

`esp_a2d_init_state_t init_state`
   a2dp profile state param

struct `a2d_psc_cfg_param`
   #include `<esp_a2dp_api.h>` ESP_A2D_SNK_PSC_CFG_EVT.

Public Members

`esp_a2d_psc_t psc_mask`
   protocol service capabilities configured
struct a2d_report_delay_stat_param
#include <esp_a2dp_api.h> ESP_A2D_REPORT_SNK_DELAY_VALUE_EVT.

Public Members

uint16_t delay_value
    delay report value

struct a2d_set_stat_param
#include <esp_a2dp_api.h> ESP_A2D_SNK_SET_DELAY_VALUE_EVT.

Public Members

esp_a2d_set_delay_value_state_t set_state
    a2dp profile state param

uint16_t delay_value
    delay report value

struct media_ctrl_stat_param
#include <esp_a2dp_api.h> ESP_A2D_MEDIA_CTRL_ACK_EVT.

Public Members

esp_a2d_media_ctrl_t cmd
    media control commands to acknowledge

esp_a2d_media_ctrl_ack_t status
    acknowledgement to media control commands

Structures

struct esp_a2d_mcc_t
    A2DP media codec capabilities union.

Public Members

esp_a2d_mct_t type
    A2DP media codec type

uint8_t sbc[ESP_A2D_CIE_LEN_SBC]
    SBC codec capabilities

uint8_t m12[ESP_A2D_CIE_LEN_M12]
    MPEG-1,2 audio codec capabilities
uint8_t m24[ESP_A2D_CIE_LEN_M24]
  MPEG-2, 4 AAC audio codec capabilities

uint8_t atrac[ESP_A2D_CIE_LEN_ATRAC]
  ATRAC family codec capabilities

union esp_a2d_mcc_t::[anonymous] cie
  A2DP codec information element

Macros

ESP_A2D_MCT_SBC
  Media codec types supported by A2DP.
  SBC

ESP_A2D_MCT_M12
  MPEG-1, 2 Audio

ESP_A2D_MCT_M24
  MPEG-2, 4 AAC

ESP_A2D_MCT_ATRAC
  ATRAC family

ESP_A2D_MCT_NON_A2DP
  NON-A2DP

ESP_A2D_PSC_DELAY_RPT
  Protocol service capabilities. This value is a mask.
  Delay Report

ESP_A2D_CIE_LEN_SBC

ESP_A2D_CIE_LEN_M12

ESP_A2D_CIE_LEN_M24

ESP_A2D_CIE_LEN_ATRAC

Type Definitions

typedef uint8_t esp_a2d_mct_t

typedef uint16_t esp_a2d_psc_t

typedef void (*esp_a2d_cb_t)(esp_a2d_cb_event_t event, esp_a2d_cb_param_t *param)
  A2DP profile callback function type.
  Param event : Event type
  Param param : Pointer to callback parameter
Chapter 2. API

typedef void (*esp_a2d_sink_data_cb_t)(const uint8_t* buf, uint32_t len)

A2DP sink data callback function.

**Param buf [in]:** pointer to the data received from A2DP source device and is PCM format decoded from SBC decoder; buf references to a static memory block and can be overwritten by upcoming data

**Param len [in]:** size(in bytes) in buf

typedef int32_t (*esp_a2d_source_data_cb_t)(uint8_t* buf, int32_t len)

A2DP source data read callback function.

**Param buf [in]:** buffer to be filled with PCM data stream from higher layer

**Param len [in]:** size(in bytes) of data block to be copied to buf. -1 is an indication to user that data buffer shall be flushed

**Return** size of bytes read successfully, if the argument len is -1, this value is ignored.

Enumerations

enum esp_a2d_connection_state_t

Bluetooth A2DP connection states.

*Values:*

enumerator ESP_A2D_CONNECTION_STATE_DISCONNECTED
connection released

enumerator ESP_A2D_CONNECTION_STATE_CONNECTING
connecting remote device

enumerator ESP_A2D_CONNECTION_STATE_CONNECTED
connection established

enumerator ESP_A2D_CONNECTION_STATE_DISCONNECTING
disconnecting remote device

enum esp_a2d_disc_rsn_t

Bluetooth A2DP disconnection reason.

*Values:*

enumerator ESP_A2D_DISC_RSN_NORMAL
Finished disconnection that is initiated by local or remote device

enumerator ESP_A2D_DISC_RSN_ABNORMAL
Abnormal disconnection caused by signal loss

enum esp_a2d_audio_state_t

Bluetooth A2DP datapath states.

*Values:*

enumerator ESP_A2D_AUDIO_STATE_REMOTE_SUSPEND
audio stream datapath suspended by remote device
Chapter 2. API

enumerator ESP_A2D_AUDIO_STATE_STOPPED
    audio stream datapath stopped

enumerator ESP_A2D_AUDIO_STATE_STARTED
    audio stream datapath started

enum esp_a2d_media_ctrl_ack_t
A2DP media control command acknowledgement code.
Values:

enumerator ESP_A2D_MEDIA_CTRL_ACK_SUCCESS
    media control command is acknowledged with success

enumerator ESP_A2D_MEDIA_CTRL_ACK_FAILURE
    media control command is acknowledged with failure

enumerator ESP_A2D_MEDIA_CTRL_ACK_BUSY
    media control command is rejected, as previous command is not yet acknowledged

enum esp_a2d_media_ctrl_t
A2DP media control commands.
Values:

enumerator ESP_A2D_MEDIA_CTRL_NONE
    Not for application use, use inside stack only.

enumerator ESP_A2D_MEDIA_CTRL_CHECK_SRC_RDY
    check whether AVDTP is connected, only used in A2DP source

enumerator ESP_A2D_MEDIA_CTRL_START
    command to set up media transmission channel

enumerator ESP_A2D_MEDIA_CTRL_STOP
    command to stop media transmission

enumerator ESP_A2D_MEDIA_CTRL_SUSPEND
    command to suspend media transmission

enum esp_a2d_init_state_t
Bluetooth A2DP Initiation states.
Values:

enumerator ESP_A2D_DEINIT_SUCCESS
    A2DP profile deinit successful event

enumerator ESP_A2D_INIT_SUCCESS
    A2DP profile deinit successful event
enum `esp_a2d_set_delay_value_state_t`

  Bluetooth A2DP set delay report value states.

  **Values:**

  - **enumerator** `ESP_A2D_SET_SUCCESS`
    
    A2DP profile set delay report value successful
  
  - **enumerator** `ESP_A2D_SET_INVALID_PARAMS`
    
    A2DP profile set delay report value is invalid parameter

enum `esp_a2d_cb_event_t`

  A2DP callback events.

  **Values:**

  - **enumerator** `ESP_A2D_CONNECTION_STATE_EVT`
    
    connection state changed event
  
  - **enumerator** `ESP_A2D_AUDIO_STATE_EVT`
    
    audio stream transmission state changed event
  
  - **enumerator** `ESP_A2D_AUDIO_CFG_EVT`
    
    audio codec is configured, only used for A2DP SINK
  
  - **enumerator** `ESP_A2D_MEDIA_CTRL_ACK_EVT`
    
    acknowledge event in response to media control commands
  
  - **enumerator** `ESP_A2D_PROF_STATE_EVT`
    
    indicate a2dp init&deinit complete
  
  - **enumerator** `ESP_A2D_SNK_PSC_CFG_EVT`
    
    protocol service capabilities configured, only used for A2DP SINK
  
  - **enumerator** `ESP_A2D_SNK_SET_DELAY_VALUE_EVT`
    
    indicate a2dp sink set delay report value complete, only used for A2DP SINK
  
  - **enumerator** `ESP_A2D_SNK_GET_DELAY_VALUE_EVT`
    
    indicate a2dp sink get delay report value complete, only used for A2DP SINK
  
  - **enumerator** `ESP_A2D_REPORT_SNK_DELAY_VALUE_EVT`
    
    report delay value, only used for A2DP SRC

**BT AVRCP APIs**

**Overview**  Bluetooth AVRCP reference APIs.

**API Reference**
**Header File**

- components/bluetooth/bluedroid/api/include/api/esp_avrc_api.h

**Functions**

`esp_err_t esp_avrc_ct_register_callback(esp_avrc_ct_cb_t callback)`

Register application callbacks to AVRCP module. This function should be called after `esp_bluedroid_enable()` completes successfully.

- 参数 `callback` : AVRCP controller callback function
  - ESP_OK: success
  - ESP_ERR_INVALID_STATE: if Bluetooth stack is not yet enabled
  - ESP_FAIL: others

`esp_err_t esp_avrc_ct_init(void)`

Initialize the Bluetooth AVRCP controller module. This function should be called after `esp_bluedroid_enable()` completes successfully. Note: AVRC cannot work independently, AVRC should be used along with A2DP and AVRC should be initialized before A2DP.

- 返回
  - ESP_OK: success
  - ESP_ERR_INVALID_STATE: if Bluetooth stack is not yet enabled
  - ESP_FAIL: others

`esp_err_t esp_avrc_ct_deinit(void)`

De-initialize AVRCP controller module. This function should be called after after `esp_bluedroid_enable()` completes successfully. Note: AVRC cannot work independently, AVRC should be used along with A2DP and AVRC should be deinitialized before A2DP.

- 返回
  - ESP_OK: success
  - ESP_ERR_INVALID_STATE: if Bluetooth stack is not yet enabled
  - ESP_FAIL: others

`esp_err_t esp_avrc_ct_send_set_player_value_cmd(uint8_t tl, uint8_t attr_id, uint8_t value_id)`

Send player application settings command to AVRCP target. This function should be called after `ESP_AVRC_CT_CONNECTION_STATE_EVT` is received and AVRC connection is established.

- 参数
  - `tl` : transaction label, 0 to 15, consecutive commands should use different values
  - `attr_id` : player application setting attribute IDs from one of `esp_avrc_ps_attr_ids_t`
  - `value_id` : attribute value defined for the specific player application setting attribute

- 返回
  - ESP_OK: success
  - ESP_ERR_INVALID_STATE: if Bluetooth stack is not yet enabled
  - ESP_FAIL: others

`esp_err_t esp_avrc_ct_send_get_rn_capabilities_cmd(uint8_t tl)`

Send GetCapabilities PDU to AVRCP target to retrieve remote device’s supported notification event_ids. This function should be called after `ESP_AVRC_CT_CONNECTION_STATE_EVT` is received and AVRC connection is established.

- 参数 `tl` : transaction label, 0 to 15, consecutive commands should use different values

- 返回
  - ESP_OK: success
  - ESP_ERR_INVALID_STATE: if Bluetooth stack is not yet enabled
  - ESP_FAIL: others
**esp_err_t esp_avrc_ct_send_register_notification_cmd** (uint8_t tl, uint8_t event_id, uint32_t event_parameter)

Send register notification command to AVRCP target. This function should be called after
ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

参数

- tl [in]: transaction label, 0 to 15, consecutive commands should use different values.
- event_id [in]: id of events, e.g. ESP_AVRC_RN_PLAY_STATUS_CHANGE, ESP_AVRC_RN_TRACK_CHANGE, etc.
- event_parameter [in]: playback interval for ESP_AVRC_RN_PLAY_POS_CHANGED; For other events, value of this parameter is ignored.

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_NOT_SUPPORTED: if the event_id is not supported in current implementation
- ESP_FAIL: others

**esp_err_t esp_avrc_ct_send_set_absolute_volume_cmd** (uint8_t tl, uint8_t volume)

Send set absolute volume command to AVRCP target. This function should be called after
ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

参数

- tl [in]: transaction label, 0 to 15, consecutive commands should use different values
- volume [in]: volume, 0 to 0x7f, means 0% to 100%

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_NOT_SUPPORTED: if the event_id is not supported in current implementation
- ESP_FAIL: others

**esp_err_t esp_avrc_ct_send_metadata_cmd** (uint8_t tl, uint8_t attr_mask)

Send metadata command to AVRCP target. This function should be called after
ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

参数

- tl [in]: transaction label, 0 to 15, consecutive commands should use different values.
- attr_mask [in]: mask of attributes, e.g. ESP_AVRC_md_ATTR_ID_TITLE | ESP_AVRC_md_ATTR_ID_ARTIST.

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_avrc_ct_send_passthrough_cmd** (uint8_t tl, uint8_t key_code, uint8_t key_state)

Send passthrough command to AVRCP target. This function should be called after
ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

参数

- tl [in]: transaction label, 0 to 15, consecutive commands should use different values.
- key_code [in]: passthrough command code, e.g. ESP_AVRC_PT_CMD_PLAY, ESP_AVRC_PT_CMD_STOP, etc.
- key_state [in]: passthrough command key state, ESP_AVRC_PT_CMD_STATE_PRESSED or ESP_AVRC_PT_CMD_STATE_RELEASED

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others
**esp_err_t esp_avrc_tg_register_callback** *(esp_avrc_tg_cb_t callback)*

Register application callbacks to AVRCP target module. This function should be called after esp_bluedroid_enable() completes successfully.

参数 **callback** [in] AVRCP target callback function

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_avrc_tg_init** *(void)*

Initialize the bluetooth AVRCP target module. This function should be called after esp_bluedroid_enable() completes successfully. Note: AVRC cannot work independently, AVRC should be used along with A2DP and AVRCP should be initialized before A2DP.

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_avrc_tg_deinit** *(void)*

De-initialize AVRCP target module. This function should be called after after esp_bluedroid_enable() completes successfully. Note: AVRC cannot work independently, AVRC should be used along with A2DP and AVRC should be deinitialized before A2DP.

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled or AVRCTG is not initialized
- ESP_ERR_INVALID_ARG: if filter type is invalid or cmd_set is NULL
- ESP_FAIL: otherwise

**esp_err_t esp_avrc_tg_get_psth_cmd_filter** *(esp_avrc_psth_filter_t filter, esp_avrc_psth_bit_mask_t *cmd_set)*

Get the current filter of remote passthrough commands on AVRC target. Filter is given by filter type and bit mask for the passthrough commands. This function should be called after esp_avrc_tg_init(). For filter type ESP_AVRC_PSTH_FILTER_ALLOWED_CMD, the retrieved command set is constant and it covers all of the passthrough commands that can possibly be supported. For filter type ESP_AVRC_PSTH_FILTER_SUPPORT_COMMANDS, the retrieved command set covers the passthrough commands selected to be supported according to current configuration. The configuration can be changed using esp_avrc_tg_set_psth_cmd_filter().

返回

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled or AVRC TG is not initialized
- ESP_ERR_INVALID_ARG: if filter type is invalid or cmd_set is NULL
- ESP_FAIL: otherwise

**esp_err_t esp_avrc_tg_set_psth_cmd_filter** *(esp_avrc_psth_filter_t filter, const esp_avrc_psth_bit_mask_t *cmd_set)*

Set the filter of remote passthrough commands on AVRC target. Filter is given by filter type and bit mask for the passthrough commands. This function should be called after esp_avrc_tg_init().

If filter type is ESP_AVRC_PSTH_FILTER_SUPPORT_CMD, the passthrough commands which are set “1” as given in cmd_set will generate ESP_AVRC_CT_PASSTHROUGH_RSP_EVT callback event and are auto-accepted in the protocol stack, other commands are replied with response type “NOT IMPLEMENTED” (8). The set of supported commands should be a subset of allowed command set. The allowed command set can be retrieved using esp_avrc_tg_get_psth_cmd_filter() with filter type “ESP_AVRC_PSTH_FILTER_ALLOWED_CMD”.

Filter type “ESP_AVRC_PSTH_FILTER_ALLOWED_CMD” does not apply to this function.

返回
• ESP_OK: success
• ESP_ERR_INVALID_STATE: if Bluetooth stack is not enabled
• ESP_ERR_INVALID_ARG: if filter type is invalid or cmd_set is NULL
• ESP_ERR_NOT_SUPPORTED: if filter type is ESP_AVRC_PSTH_FILTER_ALLOWED_CMD, or cmd_set includes commands that are not allowed

bool esp_avrc_psth_bit_mask_operation (esp_avrc_bit_mask_op_t op, esp_avrc_psth_bit_mask_t *psth, esp_avrc_pt_cmd_t cmd)

Operate on the type esp_avrc_psth_bit_mask_t with regard to a specific PASSTHROUGH command.

参数
• op [in] operation requested on the bit mask field
• psth [in] pointer to passthrough command bit mask structure
• cmd [in] passthrough command code

返回 For operation ESP_AVRC_BIT_MASK_OP_SET or ESP_AVRC_BIT_MASK_OP_CLEAR, return true for a successful operation, otherwise return false. For operation ESP_AVRC_BIT_MASK_OP_TEST, return true if the corresponding bit is set, otherwise false.

esp_err_t esp_avrc_tg_get_rn_evt_cap (esp_avrc_rn_evt_cap_t cap, esp_avrc_rn_evt_cap_mask_t *evt_set)

Get the requested event notification capabilities on local AVRC target. The capability is returned in a bit mask representation in evt_set. This function should be called after esp_avrc_tg_init().

For capability type ESP_AVRC_RN_CAP_ALLOWED_EVT, the retrieved event set is constant and it covers all of the notification events that can possibly be supported with current implementation.

For capability type ESP_AVRC_RN_CAP_SUPPORTED_EVT, the event set covers the notification events selected to be supported under current configuration. The configuration can be changed using esp_avrc_tg_set_rn_evt_cap().

返回
• ESP_OK: success
• ESP_ERR_INVALID_STATE: if Bluetooth stack is not enabled or AVRC TG is not initialized
• ESP_ERR_INVALID_ARG: if cap is invalid or evt_set is NULL
• ESP_FAIL: otherwise

esp_err_t esp_avrc_tg_set_rn_evt_cap (const esp_avrc_rn_evt_cap_mask_t *evt_set)

Set the event notification capabilities on local AVRCP target. The capability is given in a bit mask representation in evt_set and must be a subset of allowed event IDs with current implementation. This function should be called after esp_avrc_tg_init().

返回
• ESP_OK: success
• ESP_ERR_INVALID_STATE: if Bluetooth stack is not enabled
• ESP_ERR_INVALID_ARG: if evt_set is NULL

bool esp_avrc_rn_evt_bit_mask_operation (esp_avrc_bit_mask_op_t op, esp_avrc_rn_evt_cap_mask_t *events, esp_avrc_rn_event_ids_t event_id)

Operate on the type esp_avrc_rn_evt_cap_mask_t with regard to a specific event.

For operation ESP_AVRC_BIT_MASK_OP_TEST, return true if the corresponding bit is set, otherwise false.

参数
• op [in] operation requested on the bit mask field
• events [in] pointer to event notification capability bit mask structure
• event_id [in] notification event code
For operation ESP_AVRC_BIT_MASK_OP_SET or ESP_AVRC_BIT_MASK_OP_CLEAR, return true for a successful operation, otherwise return false.

```c
esp_err_t esp_avrc_tg_send_rn_rsp(esp_avrc_rn_event_ids_t event_id, esp_avrc_rn_rsp_t rsp,
                                    esp_avrc_rn_param_t *param)
```

Send RegisterNotification Response to remote AVRCP controller. Local event notification capability can be set using esp_avrc_tg_set_rn_evt_cap(), in a bit mask representation in evt_set. This function should be called after esp_avrc_tg_init().

### Parameters
- `event_id` - [in] notification event ID that remote AVRCP CT registers
- `rsp` - [in] notification response code
- `param` - [in] parameters included in the specific notification

### Return
- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled or AVRC TG is not initialized
- ESP_ERR_INVALID_ARG: if evt_set is NULL

### Unions

```c
union esp_avrc_rn_param_t
```

```
#include <esp_avrc_api.h>
```

AVRCP notification parameters.

#### Public Members

- `uint8_t volume`
  - response data for ESP_AVRC_RN_VOLUME_CHANGE, ranges 0..127

- `esp_avrc_playback_stat_t playback`
  - response data for ESP_AVRC_RN_PLAY_STATUS_CHANGE

- `uint8_t elm_id[8]`
  - response data for ESP_AVRC_RN_TRACK_CHANGE

- `uint32_t play_pos`
  - response data for ESP_AVRC_RN_PLAY_POS_CHANGED, in millisecond

- `esp_avrc_batt_stat_t batt`
  - response data for ESP_AVRC_RN_BATTERY_STATUS_CHANGE

### Unions

```c
union esp_avrc_ct_cb_param_t
```

```
#include <esp_avrc_api.h>
```

AVRC controller callback parameters.

#### Public Members

- `struct esp_avrc_ct_cb_param_t::avrc_ct_conn_stat_param conn_stat`
  - AVRC connection status

- `struct esp_avrc_ct_cb_param_t::avrc_ct_psth_rsp_param psth_rsp`
  - passthrough command response
struct esp_avrc_ct_cb_param_t::avrc_ct_meta_rsp_param meta_rsp
    metadata attributes response

struct esp_avrc_ct_cb_param_t::avrc_ct_change_notify_param change_ntf
    notifications

struct esp_avrc_ct_cb_param_t::avrc_ct_rmt_feats_param rmt_feats
    AVRC features discovered from remote SDP server

struct esp_avrc_ct_cb_param_t::avrc_ct_get_rn_caps_rsp_param get_rn_caps_rsp
    get supported event capabilities response from AVRCP target

struct esp_avrc_ct_cb_param_t::avrc_ct_set_volume_rsp_param set_volume_rsp
    set absolute volume response event

struct avrc_ct_change_notify_param
    #include <esp_avrc_api.h> ESP_AVRC_CT_CHANGE_NOTIFY_EVT.

Public Members

    uint8_t event_id
        id of AVRC event notification

    esp_avrc_rn_param_t event_parameter
        event notification parameter

struct avrc_ct_conn_stat_param
    #include <esp_avrc_api.h> ESP_AVRC_CT_CONNECTION_STATE_EVT.

Public Members

    bool connected
        whether AVRC connection is set up

    esp_bd_addr_t remote_bda
        remote bluetooth device address

struct avrc_ct_get_rn_caps_rsp_param
    #include <esp_avrc_api.h> ESP_AVRC_CT_GET_RN_CAPABILITIES_RSP_EVT.

Public Members

    uint8_t cap_count
        number of items provided in event or company_id according to cap_id used
struct esp_avrc_rn_evt_cap_mask_t { 
   (evt_set
        supported event_ids represented in bit-mask 
    
    struct avrc_ct_meta_rsp_param 
        #include <esp_avrc_api.h> ESP_AVRC_CT_METADATA_RSP_EVT. 
    
        Public Members 
    
    uint8_t attr_id 
        id of metadata attribute 
    
    uint8_t* attr_text 
        attribute itself 
    
    int attr_length 
        attribute character length 
    
    struct avrc_ct_psth_rsp_param 
        #include <esp_avrc_api.h> ESP_AVRC_CT_PASSTHROUGH_RSP_EVT. 
    
        Public Members 
    
    uint8_t tl 
        transaction label, 0 to 15 
    
    uint8_t key_code 
        passthrough command code 
    
    uint8_t key_state 
        0 for PRESSED, 1 for RELEASED 
    
    struct avrc_ct_rmt_feats_param 
        #include <esp_avrc_api.h> ESP_AVRC_CT_REMOTE_FEATURES_EVT. 
    
        Public Members 
    
    uint32_t feat_mask 
        AVRC feature mask of remote device 
    
    uint16_t tg_feat_flag 
        feature flag of remote device as TG 
    
    esp_bd_addr_t remote_bda 
        remote bluetooth device address 
    
    struct avrc_ct_set_volume_rsp_param 
        #include <esp_avrc_api.h> ESP_AVRC_CT_SET_ABSOLUTE_VOLUME_RSP_EVT. 
    
        esp_avrc_rn_evt_cap_mask_t
Chapter 2. API

Public Members

```c
uint8_t volume
```
the volume which has actually been set, range is 0 to 0x7f, means 0% to 100%

```c
union esp_avrc_tg_cb_param_t
```
#include <esp_avrc_api.h> AVRC target callback parameters.

Public Members

```c
struct esp_avrc_tg_cb_param_t::avrc_tg_conn_stat_param conn_stat
```
AVRC connection status

```c
struct esp_avrc_tg_cb_param_t::avrc_tg_rmt_feats_param rmt_feats
```
AVRC features discovered through SDP

```c
struct esp_avrc_tg_cb_param_t::avrc_tg_psth_cmd_param psth_cmd
```
passthrough command

```c
struct esp_avrc_tg_cb_param_t::avrc_tg_set_abs_vol_param set_abs_vol
```
set absolute volume command targeted on audio sink

```c
struct esp_avrc_tg_cb_param_t::avrc_tg_reg_ntf_param reg_ntf
```
register notification

```c
struct esp_avrc_tg_cb_param_t::avrc_tg_set_app_value_param set_app_value
```
set player application value

```c
struct avrc_tg_conn_stat_param
```
#include <esp_avrc_api.h> ESP_AVRC_TG_CONNECTION_STATE_EVT.

Public Members

```c
bool connected
```
whether AVRC connection is set up

```c
esp_bd_addr_t remote_bda
```
remote bluetooth device address

```c
struct avrc_tg_psth_cmd_param
```
#include <esp_avrc_api.h> ESP_AVRC_TG_PASSTHROUGH_CMD_EVT.

Public Members

```c
uint8_t key_code
```
passthrough command code
uint8_t key_state
   0 for Pressed, 1 for Released

struct avrc_tg_reg_ntf_param
#include <esp_avrc_api.h> ESP_AVRC_TG_REGISTER_NOTIFICATION_EVT.

Public Members

uint8_t event_id
   event id of AVRC RegisterNotification

uint32_t event_parameter
   event notification parameter

struct avrc_tg_rmt_feats_param
#include <esp_avrc_api.h> ESP_AVRC_TG_REMOTE_FEATURES_EVT.

Public Members

uint32_t feat_mask
   AVRC feature mask of remote device

uint16_t ct_feat_flag
   feature flag of remote device as CT

esp_bd_addr_t remote_bda
   remote bluetooth device address

struct avrc_tg_set_abs_vol_param
#include <esp_avrc_api.h> ESP_AVRC_TG_SET_ABSOLUTE_VOLUME_CMD_EVT.

Public Members

uint8_t volume
   volume ranges from 0 to 127

struct avrc_tg_set_app_value_param
#include <esp_avrc_api.h> ESP_AVRC_TG_SET_PLAYER_APP_VALUE_EVT.

Public Members

uint8_t num_val
   attribute num

esp_avrc_set_app_value_param_t *p_vals
   point to the id and value of player application attribute
Chapter 2. API Reference

Structures

struct esp_avrc_psth_bit_mask_t
   AVRC passthrough command bit mask.

Public Members

uint16_t bits[8]
   bit mask representation of PASSTHROUGH commands

struct esp_avrc_rn_evt_cap_mask_t
   AVRC target notification event capability bit mask.

Public Members

uint16_t bits
   bit mask representation of PASSTHROUGH commands

struct esp_avrc_set_app_value_param_t
   AVRCP set app value parameters.

Public Members

uint8_t attr_id
   player application attribute id

uint8_t attr_val
   player application attribute value

Macros

ESP_AVRC_TRANS_LABEL_MAX
   max transaction label

Type Definitions

typedef void (*esp_avrc_ct_cb_t)(esp_avrc_ct_cb_event_t event, esp_avrc_ct_cb_param_t *param)
   AVRCP controller callback function type.
   Param event : Event type
   Param param : Pointer to callback parameter union

typedef void (*esp_avrc_tg_cb_t)(esp_avrc_tg_cb_event_t event, esp_avrc_tg_cb_param_t *param)
   AVRCP target callback function type.
   Param event : Event type
   Param param : Pointer to callback parameter union
Enumerations

**enum esp_avrc_features_t**

AVRC feature bit mask.

*Values:*

- enumerator **ESP_AVRC_FEAT_RCTG**
  - remote control target

- enumerator **ESP_AVRC_FEAT_RCCT**
  - remote control controller

- enumerator **ESP_AVRC_FEAT_VENDOR**
  - remote control vendor dependent commands

- enumerator **ESP_AVRC_FEAT_BROWSE**
  - use browsing channel

- enumerator **ESP_AVRC_FEAT_META_DATA**
  - remote control metadata transfer command/response

- enumerator **ESP_AVRC_FEAT_ADV_CTRL**
  - remote control advanced control command/response

**enum esp_avrc_feature_flag_t**

AVRC supported features flag retrieved in SDP record.

*Values:*

- enumerator **ESP_AVRC_FEAT_FLAG_CAT1**
  - category 1

- enumerator **ESP_AVRC_FEAT_FLAG_CAT2**
  - category 2

- enumerator **ESP_AVRC_FEAT_FLAG_CAT3**
  - category 3

- enumerator **ESP_AVRC_FEAT_FLAG_CAT4**
  - category 4

- enumerator **ESP_AVRC_FEAT_FLAG_BROWSING**
  - browsing

- enumerator **ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE_PROP**
  - Cover Art GetImageProperties

- enumerator **ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE**
  - Cover Art GetImage
Chapter 2. API 参考

enumerator `ESP_AVRC_FEAT_FLAG_COVER_ART_GET_LINKED_THUMBNAIL`
  Cover Art GetLinkedThumbnail

data type `esp_avrc_pt_cmd_t`
  AVRC passthrough command code.
  
  Values:

  enumerator `ESP_AVRC_PT_CMD_SELECT`
    select
  enumerator `ESP_AVRC_PT_CMD_UP`
    up
  enumerator `ESP_AVRC_PT_CMD_DOWN`
    down
  enumerator `ESP_AVRC_PT_CMD_LEFT`
    left
  enumerator `ESP_AVRC_PT_CMD_RIGHT`
    right
  enumerator `ESP_AVRC_PT_CMD_RIGHT_UP`
    right-up
  enumerator `ESP_AVRC_PT_CMD_RIGHT_DOWN`
    right-down
  enumerator `ESP_AVRC_PT_CMD_LEFT_UP`
    left-up
  enumerator `ESP_AVRC_PT_CMD_LEFT_DOWN`
    left-down
  enumerator `ESP_AVRC_PT_CMD_ROOT_MENU`
    root menu
  enumerator `ESP_AVRC_PT_CMD_SETUP_MENU`
    setup menu
  enumerator `ESP_AVRC_PT_CMD_CONT_MENU`
    contents menu
  enumerator `ESP_AVRC_PT_CMD_FAV_MENU`
    favorite menu
  enumerator `ESP_AVRC_PT_CMD_EXIT`
    exit
enumerator ESP_AVRC_PT_CMD_0
  0
enumerator ESP_AVRC_PT_CMD_1
  1
enumerator ESP_AVRC_PT_CMD_2
  2
enumerator ESP_AVRC_PT_CMD_3
  3
enumerator ESP_AVRC_PT_CMD_4
  4
enumerator ESP_AVRC_PT_CMD_5
  5
enumerator ESP_AVRC_PT_CMD_6
  6
enumerator ESP_AVRC_PT_CMD_7
  7
enumerator ESP_AVRC_PT_CMD_8
  8
enumerator ESP_AVRC_PT_CMD_9
  9
enumerator ESP_AVRC_PT_CMD_DOT
dot
enumerator ESP_AVRC_PT_CMD_ENTER
enter
enumerator ESP_AVRC_PT_CMD_CLEAR
clear
enumerator ESP_AVRC_PT_CMD_CHAN_UP
channel up
enumerator ESP_AVRC_PT_CMD_CHAN_DOWN
channel down
enumerator ESP_AVRC_PT_CMD_PREV_CHAN
previous channel
enumerator `ESP_AVRC_PT_CMD_SOUND_SEL`  
sound select

distributor `ESP_AVRC_PT_CMD_INPUT_SEL`  
input select

distributor `ESP_AVRC_PT_CMD_DISP_INFO`  
display information

distributor `ESP_AVRC_PT_CMD_HELP`  
help

distributor `ESP_AVRC_PT_CMD_PAGE_UP`  
page up

distributor `ESP_AVRC_PT_CMD_PAGE_DOWN`  
page down

distributor `ESP_AVRC_PT_CMD_POWER`  
power

distributor `ESP_AVRC_PT_CMD_VOL_UP`  
volume up

distributor `ESP_AVRC_PT_CMD_VOL_DOWN`  
volume down

distributor `ESP_AVRC_PT_CMD_MUTE`  
mute

distributor `ESP_AVRC_PT_CMD_PLAY`  
play

distributor `ESP_AVRC_PT_CMD_STOP`  
stop

distributor `ESP_AVRC_PT_CMD_PAUSE`  
pause

distributor `ESP_AVRC_PT_CMD_RECORD`  
record

distributor `ESP_AVRC_PT_CMD_REWIND`  
rewind

distributor `ESP_AVRC_PT_CMD_FAST_FORWARD`  
fast forward
enumerator ESP_AVRC_PT_CMD_EJECT
eject
enumerator ESP_AVRC_PT_CMD_FORWARD
forward
enumerator ESP_AVRC_PT_CMD_BACKWARD
backward
enumerator ESP_AVRC_PT_CMD_ANGLE
angle
enumerator ESP_AVRC_PT_CMD_SUBPICT
subpicture
enumerator ESP_AVRC_PT_CMD_F1
F1
enumerator ESP_AVRC_PT_CMD_F2
F2
enumerator ESP_AVRC_PT_CMD_F3
F3
enumerator ESP_AVRC_PT_CMD_F4
F4
enumerator ESP_AVRC_PT_CMD_F5
F5
enumerator ESP_AVRC_PT_CMD_VENDOR
vendor unique

data type esp_avrc_psth_filter_t
AVRC passthrough command filter.
Values:

enumerator ESP_AVRC_PSTH_FILTER_ALLOWED_CMD
all of the PASSTHROUGH commands that can possibly be used, immutable
enumerator ESP_AVRC_PSTH_FILTER_SUPPORTED_CMD
PASSTHROUGH commands selectively supported according to the current configuration
enumerator ESP_AVRC_PSTH_FILTER_SUPPORT_MAX


data type esp_avrc_bit_mask_op_t
Values:
enumerator `ESP_AVRC_BIT_MASK_OP_TEST`
   operation code to test a specific bit

enumerator `ESP_AVRC_BIT_MASK_OP_SET`
   operation code to set a specific bit

enumerator `ESP_AVRC_BIT_MASK_OP_CLEAR`
   operation code to clear a specific bit

enum `esp_avrc_pt_cmd_state_t`
AVRC passthrough command state.
   Values:
   
enumerator `ESP_AVRC_PT_CMD_STATE_PRESSED`
   key pressed

enumerator `ESP_AVRC_PT_CMD_STATE_RELEASED`
   key released

enum `esp_avrc_ct_cb_event_t`
AVRC Controller callback events.
   Values:
   
enumerator `ESP_AVRC_CT_CONNECTION_STATE_EVT`
   connection state changed event

enumerator `ESP_AVRC_CT_PASSTHROUGH_RSP_EVT`
   passthrough response event

enumerator `ESP_AVRC_CT_METADATA_RSP_EVT`
   metadata response event

enumerator `ESP_AVRC_CT_PLAY_STATUS_RSP_EVT`
   play status response event

enumerator `ESP_AVRC_CT_CHANGE_NOTIFY_EVT`
   notification event

enumerator `ESP_AVRC_CT_REMOTE_FEATURES_EVT`
   feature of remote device indication event

enumerator `ESP_AVRC_CT_GET_RN_CAPABILITIES_RSP_EVT`
   supported notification events capability of peer device

enumerator `ESP_AVRC_CT_SET_ABSOLUTE_VOLUME_RSP_EVT`
   set absolute volume response event
enum esp_avrc_tg_cb_event_t
AVRC Target callback events.

Values:

- enumerator ESP_AVRC_TG_CONNECTION_STATE_EVT
  connection state changed event

- enumerator ESP_AVRC_TG_REMOTE_FEATURES_EVT
  feature of remote device indication event

- enumerator ESP_AVRC_TG_PASSTHROUGH_CMD_EVT
  passthrough command event

- enumerator ESP_AVRC_TG_SET_ABSOLUTE_VOLUME_CMD_EVT
  set absolute volume command from remote device

- enumerator ESP_AVRC_TG_REGISTER_NOTIFICATION_EVT
  register notification event

- enumerator ESP_AVRC_TG_SET_PLAYER_APP_VALUE_EVT
  set application attribute value, attribute refer to esp_avrc_ps_attr_ids_t

enum esp_avrc_md_attr_mask_t
AVRC metadata attribute mask.

Values:

- enumerator ESP_AVRC_MD_ATTR_TITLE
  title of the playing track

- enumerator ESP_AVRC_MD_ATTR_ARTIST
  track artist

- enumerator ESP_AVRC_MD_ATTR_ALBUM
  album name

- enumerator ESP_AVRC_MD_ATTR_TRACK_NUM
  track position on the album

- enumerator ESP_AVRC_MD_ATTR_NUM_TRACKS
  number of tracks on the album

- enumerator ESP_AVRC_MD_ATTR_GENRE
  track genre

- enumerator ESP_AVRC_MD_ATTR_PLAYING_TIME
  total album playing time in milliseconds
enum esp_avrc_rn_event_ids_t
AVRC event notification ids.

Values:

enumerator ESP_AVRC_RN_PLAY_STATUS_CHANGE
   track status change, eg. from playing to paused

enumerator ESP_AVRC_RN_TRACK_CHANGE
   new track is loaded

enumerator ESP_AVRC_RN_TRACK_REACHED_END
   current track reached end

enumerator ESP_AVRC_RN_TRACK_REACHED_START
   current track reached start position

enumerator ESP_AVRC_RN_PLAY_POS_CHANGED
   track playing position changed

enumerator ESP_AVRC_RN_BATTERY_STATUS_CHANGE
   battery status changed

enumerator ESP_AVRC_RN_SYSTEM_STATUS_CHANGE
   system status changed

enumerator ESP_AVRC_RN_APP_SETTING_CHANGE
   application settings changed

enumerator ESP_AVRC_RN_NOW_PLAYING_CHANGE
   now playing content changed

enumerator ESP_AVRC_RN_AVAILABLE_PLAYERS_CHANGE
   available players changed

enumerator ESP_AVRC_RN_ADDRESSED_PLAYER_CHANGE
   the addressed player changed

enumerator ESP_AVRC_RN_UIDS_CHANGE
   UIDs changed

enumerator ESP_AVRC_RN_VOLUME_CHANGE
   volume changed locally on TG

enumerator ESP_AVRC_RN_MAX_EVT
enumerator **ESP_AVRC_RN_CAP_ALLOWED_EVT**
all of the notification events that can possibly be supported, immutable

enumerator **ESP_AVRC_RN_CAP_SUPPORTED_EVT**
note that notification events selectively supported according to the current configuration

enumerator **ESP_AVRC_RN_CAP_MAX**

enum **esp_avrc_rn_rsp_t**
AVRC notification response type.

*Values:*

enumerator **ESP_AVRC_RN_RSP_INTERIM**
initial response to RegisterNotification, should be sent T_mtp(1000ms) from receiving the command

enumerator **ESP_AVRC_RN_RSP_CHANGED**
final response to RegisterNotification command

enum **esp_avrc_ps_attr_ids_t**
AVRC player setting ids.

*Values:*

enumerator **ESP_AVRC_PS_EQUALIZER**
equalizer, on or off

enumerator **ESP_AVRC_PS_REPEAT_MODE**
repeat mode

enumerator **ESP_AVRC_PS_SHUFFLE_MODE**
shuffle mode

enumerator **ESP_AVRC_PS_SCAN_MODE**
scan mode on or off

enumerator **ESP_AVRC_PS_MAX_ATTR**

enum **esp_avrc_ps_eq_value_ids_t**
AVRC equalizer modes.

*Values:*

enumerator **ESP_AVRC_PS_EQUALIZER_OFF**
equalizer OFF

enumerator **ESP_AVRC_PS_EQUALIZER_ON**
equalizer ON
enum `esp_avrc_ps_rpt_value_ids_t`
  AVRC repeat modes.
  Values:

  enumerator `ESP_AVRC_PS_REPEAT_OFF`
  repeat mode off

  enumerator `ESP_AVRC_PS_REPEAT_SINGLE`
  single track repeat

  enumerator `ESP_AVRC_PS_REPEAT_GROUP`
  group repeat

enum `esp_avrc_ps_shf_value_ids_t`
  AVRC shuffle modes.
  Values:

  enumerator `ESP_AVRC_PS_SHUFFLE_OFF`

  enumerator `ESP_AVRC_PS_SHUFFLE_ALL`

  enumerator `ESP_AVRC_PS_SHUFFLE_GROUP`

enum `esp_avrc_ps_scn_value_ids_t`
  AVRC scan modes.
  Values:

  enumerator `ESP_AVRC_PS_SCAN_OFF`
  scan off

  enumerator `ESP_AVRC_PS_SCAN_ALL`
  all tracks scan

  enumerator `ESP_AVRC_PS_SCAN_GROUP`
  group scan

enum `esp_avrc_rsp_t`
  AVCTP response codes.
  Values:

  enumerator `ESP_AVRC_RSP_NOT_IMPL`
  not implemented

  enumerator `ESP_AVRC_RSP_ACCEPT`
  accept
enumerator ESP_AVRC_RSP_REJECT
    reject
enumerator ESP_AVRC_RSP_IN_TRANS
    in transition
enumerator ESP_AVRC_RSP_IMPL_STBL
    implemented/stable
enumerator ESP_AVRC_RSP_CHANGED
    changed
enumerator ESP_AVRC_RSP_INTERIM
    interim

enum esp_avrc_batt_stat_t
    AVRCP battery status.
    Values:
    enumerator ESP_AVRC_BATT_NORMAL
        normal state
    enumerator ESP_AVRC_BATT_WARNING
        unable to operate soon
    enumerator ESP_AVRC_BATT_CRITICAL
        cannot operate any more
    enumerator ESP_AVRC_BATT_EXTERNAL
        plugged to external power supply
    enumerator ESP_AVRC_BATT_FULL_CHARGE
        when completely charged from external power supply

enum esp_avrc_playback_stat_t
    AVRCP current status of playback.
    Values:
    enumerator ESP_AVRC_PLAYBACK_STOPPED
        stopped
    enumerator ESP_AVRC_PLAYBACK_PLAYING
        playing
    enumerator ESP_AVRC_PLAYBACK_PAUSED
        paused
enumerator **ESP_AVRC_PLAYBACK_FWD_SEEK**
   forward seek

enumerator **ESP_AVRC_PLAYBACK_REV_SEEK**
   reverse seek

enumerator **ESP_AVRC_PLAYBACK_ERROR**
   error

**SPP API**

**Application Example**  Check bluetooth/bluedroid/classic_bt folder in ESP-IDF examples, which contains the following application:

- This is a SPP demo. This demo can discover the service, connect, send and receive SPP data bluetooth/bluedroid/classic_bt/bt_spp_acceptor, bluetooth/bluedroid/classic_bt/bt_spp_initiator

**API Reference**

**Header File**

- components/bluetooth/bluedroid/api/include/api/esp_spp_api.h

**Functions**

```c
esp_err_t esp_spp_register_callback(esp_spp_cb_t callback)
```

This function is called to init callbacks with SPP module.

- **callback** [in] pointer to the init callback function.
  - ESP_OK: success
  - other: failed

```c
esp_err_t esp_spp_init(esp_spp_mode_t mode)
```

This function is called to init SPP module. When the operation is completed, the callback function will be called with ESP_SPP_INIT_EVT. This function should be called after esp_bluedroid_enable() completes successfully.

- **mode** [in] Choose the mode of SPP, ESP_SPP_MODE_CB or ESP_SPP_MODE_VFS.
  - ESP_OK: success
  - other: failed

```c
esp_err_t esp_spp_enhanced_init(const esp_spp_cfg_t *cfg)
```

This function is called to init SPP module. When the operation is completed, the callback function will be called with ESP_SPP_INIT_EVT. This function should be called after esp_bluedroid_enable() completes successfully.

- **cfg** [in] SPP configuration.
  - ESP_OK: success
  - other: failed

---

Espressif Systems 415  Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
**esp_err_t esp_spp_deinit** (void)

This function is called to uninit SPP module. The operation will close all active SPP connection first, then the callback function will be called with ESP_SPP_CLOSE_EVT, and the number of ESP_SPP_CLOSE_EVT is equal to the number of connection. When the operation is completed, the callback function will be called with ESP_SPP_UNINIT_EVT. This function should be called after esp_spp_init()/esp_spp_enhanced_init() completes successfully.

- **ESP_OK**: success
- **other**: failed

**esp_err_t esp_spp_start_discovery** (esp_bd_addr_t bd_addr)

This function is called to perform service discovery for the services provided by the given peer device. When the operation is completed, the callback function will be called with ESP_SPP_DISCOVERY_COMP_EVT. This function must be called after esp_spp_init()/esp_spp_enhanced_init() successful and before esp_spp_deinit().

参数 **bd_addr** - [in] Remote device bluetooth device address.

- **ESP_OK**: success
- **other**: failed

**esp_err_t esp_spp_connect** (esp_spp_sec_t sec_mask, esp_spp_role_t role, uint8_t remote_scn, esp_bd_addr_t peer_bd_addr)

This function makes an SPP connection to a remote BD Address. When the connection is initiated or failed to initiate, the callback is called with ESP_SPP_CL_INIT_EVT. When the connection is established or failed, the callback is called with ESP_SPP_OPEN_EVT. This function must be called after esp_spp_init()/esp_spp_enhanced_init() successful and before esp_spp_deinit().

参数  
- **role** - [in] Master or slave.
- **remote_scn** - [in] Remote device bluetooth device SCN.
- **peer_bd_addr** - [in] Remote device bluetooth device address.

返回  
- **ESP_OK**: success
- **other**: failed

**esp_err_t esp_spp_disconnect** (uint32_t handle)

This function closes an SPP connection. When the operation is completed, the callback function will be called with ESP_SPP_CLOSE_EVT. This function must be called after esp_spp_init()/esp_spp_enhanced_init() successful and before esp_spp_deinit().

参数 **handle** - [in] The connection handle.

返回  
- **ESP_OK**: success
- **other**: failed

**esp_err_t esp_spp_start_srv** (esp_spp_sec_t sec_mask, esp_spp_role_t role, uint8_t local_scn, const char *name)

This function create a SPP server and starts listening for an SPP connection request from a remote Bluetooth device. When the server is started successfully, the callback is called with ESP_SPP_START_EVT. When the connection is established, the callback is called with ESP_SPP_SRV_OPEN_EVT. This function must be called after esp_spp_init()/esp_spp_enhanced_init() successful and before esp_spp_deinit().

参数  
- **role** - [in] Master or slave.
Chapter 2. API 参考

- **local_scn** - [in] The specific channel you want to get. If channel is 0, means get any channel.
- **name** - [in] Server’s name.

返回
- ESP_OK: success
- other: failed

```c
 esp_err_t esp_spp_stop_srv (void)
```

This function stops all SPP servers. The operation will close all active SPP connection first, then the callback function will be called with ESP_SPP_CLOSE_EVT, and the number of ESP_SPP_CLOSE_EVT is equal to the number of connection. When the operation is completed, the callback is called with ESP_SPP_SRV_STOP_EVT. This function must be called after esp_spp_init()/esp_spp_enhanced_init() successful and before esp_spp_deinit().

返回
- ESP_OK: success
- other: failed

```c
 esp_err_t esp_spp_stop_srv_scn (uint8_t scn)
```

This function stops a specific SPP server. The operation will close all active SPP connection first on the specific SPP server, then the callback function will be called with ESP_SPP_CLOSE_EVT, and the number of ESP_SPP_CLOSE_EVT is equal to the number of connection. When the operation is completed, the callback is called with ESP_SPP_SRV_STOP_EVT. This function must be called after esp_spp_init()/esp_spp_enhanced_init() successful and before esp_spp_deinit().

参数
- **scn** - [in] Server channel number.

返回
- ESP_OK: success
- other: failed

```c
 esp_err_t esp_spp_write (uint32_t handle, int len, uint8_t *p_data)
```

This function is used to write data, only for ESP_SPP_MODE_CB. When this function need to be called repeatedly, it is strongly recommended to call this function again after the previous event ESP_SPP_WRITE_EVT is received and the parameter ‘cong’ is equal to false. If the previous event ESP_SPP_WRITE_EVT with parameter ‘cong’ is equal to true, the function can only be called again when the event ESP_SPP_CONG_EVT with parameter ‘cong’ equal to false is received. This function must be called after an connection between initiator and acceptor has been established.

参数
- **handle** - [in] The connection handle.
- **len** - [in] The length of the data written.

返回
- ESP_OK: success
- other: failed

```c
 esp_err_t esp_spp_vfs_register (void)
```

This function is used to register VFS. For now, SPP only supports write, read and close.

返回
- ESP_OK: success
- other: failed

Unions

```c
 union esp_spp_cb_param_t
```

SPP callback parameters union.

Public Members
struct esp_spp_cb_param_t::spp_init_evt_param init
    SPP callback param of SPP_INIT_EVT

struct esp_spp_cb_param_t::spp_uninit_evt_param uninit
    SPP callback param of SPP_UNINIT_EVT

struct esp_spp_cb_param_t::spp_discovery_comp_evt_param disc_comp
    SPP callback param of SPP_DISCOVERY_COMP_EVT

struct esp_spp_cb_param_t::spp_open_evt_param open
    SPP callback param of ESP_SPP_OPEN_EVT

struct esp_spp_cb_param_t::spp_srv_open_evt_param srv_open
    SPP callback param of ESP_SPP_SRV_OPEN_EVT

struct esp_spp_cb_param_t::spp_close_evt_param close
    SPP callback param of ESP_SPP_CLOSE_EVT

struct esp_spp_cb_param_t::spp_start_evt_param start
    SPP callback param of ESP_SPP_START_EVT

struct esp_spp_cb_param_t::spp_srv_stop_evt_param srv_stop
    SPP callback param of ESP_SPP_SRV_STOP_EVT

struct esp_spp_cb_param_t::spp_cl_init_evt_param cl_init
    SPP callback param of ESP_SPP_CL_INIT_EVT

struct esp_spp_cb_param_t::spp_write_evt_param write
    SPP callback param of ESP_SPP_WRITE_EVT

struct esp_spp_cb_param_t::spp_data_ind_evt_param data_ind
    SPP callback param of ESP_SPP_DATA_IND_EVT

struct esp_spp_cb_param_t::spp_cong_evt_param cong
    SPP callback param of ESP_SPP_CONG_EVT

struct spp_cl_init_evt_param
    #include <esp_spp_api.h> ESP_SPP_CL_INIT_EVT.

Public Members

esp_spp_status_t status
    status

uint32_t handle
    The connection handle
uint8_t sec_id
    security ID used by this server

bool use_co
    TRUE to use co_rfc_data

struct spp_close_evt_param
#include <esp_spp_api.h> ESP_SPP_CLOSE_EVT.

Public Members

esp_spp_status_t status
    status

uint32_t port_status
    PORT status

uint32_t handle
    The connection handle

bool async
    FALSE, if local initiates disconnect

struct spp_cong_evt_param
#include <esp_spp_api.h> ESP_SPP_CONG_EVT.

Public Members

esp_spp_status_t status
    status

uint32_t handle
    The connection handle

bool cong
    TRUE, congested. FALSE, uncongested

struct spp_data_ind_evt_param
#include <esp_spp_api.h> ESP_SPP_DATA_IND_EVT.

Public Members

esp_spp_status_t status
    status
Chapter 2. API 参考

```c
uint32_t handle
    The connection handle

uint16_t len
    The length of data

uint8_t *data
    The data received
```

### struct `spp_discovery_comp_evt_param`

```
#include <esp_spp_api.h> SPP_DISCOVERY_COMP_EVT.
```

#### Public Members

```c
esp_spp_status_t status
    status

uint8_t scn_num
    The num of scn_num

uint8_t *sccn[ESP_SPP_MAX_SCN]
    channel #

const char *service_name[ESP_SPP_MAX_SCN]
    service_name
```

### struct `spp_init_evt_param`

```
#include <esp_spp_api.h> SPP_INIT_EVT.
```

#### Public Members

```c
esp_spp_status_t status
    status
```

### struct `spp_open_evt_param`

```
#include <esp_spp_api.h> ESP_SPP_OPEN_EVT.
```

#### Public Members

```c
esp_spp_status_t status
    status

uint32_t handle
    The connection handle
```
int fd
    The file descriptor only for ESP_SPP_MODE_VFS

    esp_bd_addr_t rem_bda
    The peer address

struct spp_srv_open_evt_param
    #include <esp_spp_api.h> ESP_SPP_SRV_OPEN_EVT.

    Public Members

    esp_spp_status_t status
        status

    uint32_t handle
        The connection handle

    uint32_t new_listen_handle
        The new listen handle

    int fd
        The file descriptor only for ESP_SPP_MODE_VFS

    esp_bd_addr_t rem_bda
        The peer address

struct spp_srv_stop_evt_param
    #include <esp_spp_api.h> ESP_SPP_SRV_STOP_EVT.

    Public Members

    esp_spp_status_t status
        status

    uint8_t scn
        Server channel number

struct spp_start_evt_param
    #include <esp_spp_api.h> ESP_SPP_START_EVT.

    Public Members

    esp_spp_status_t status
        status
Chapter 2. API 参考

```c
uint32_t handle
   The connection handle

uint8_t sec_id
   security ID used by this server

uint8_t scn
   Server channel number

bool use_co
   TRUE to use co_rfc_data

struct spp_uninit_evt_param
   #include <esp_spp_api.h> SPP_UNINIT_EVT.

  Public Members

  esp_spp_status_t status

struct spp_write_evt_param
   #include <esp_spp_api.h> ESP_SPP_WRITE_EVT.

  Public Members

  esp_spp_status_t status

  uint32_t handle
     The connection handle

  int len
     The length of the data written.

  bool cong
     congestion status

Structures

struct esp_spp_cfg_t
   SPP configuration parameters.

  Public Members

  esp_spp_mode_t mode
     Choose the mode of SPP, ESP_SPP_MODE_CB or ESP_SPP_MODE_VFS.
```
bool enable_l2cap_ertm
   Enable/disable Logical Link Control and Adaptation Layer Protocol enhanced retransmission mode.

uint16_t tx_buffer_size
   Tx buffer size for a new SPP channel. A smaller setting can save memory, but may incur a decrease in throughput. Only for ESP_SPP_MODE_VFS mode.

**Macros**

**ESP_SPP_MAX_MTU**
   SPP max MTU

**ESP_SPP_MAX_SCN**
   SPP max SCN

**ESP_SPP_MIN_TX_BUFFER_SIZE**
   SPP min tx buffer

**ESP_SPP_MAX_TX_BUFFER_SIZE**
   SPP max tx buffer size

**BT_SPP_DEFAULT_CONFIG ()**
   SPP default configuration.

**ESP_SPP_SEC_NONE**
   No security. relate to BTA_SEC_NONE in bta/bta_api.h

**ESP_SPP_SEC_AUTHORIZE**
   Authorization required (only needed for out going connection ) relate to BTA_SEC_AUTHORIZE in bta/bta_api.h

**ESP_SPP_SEC_AUTHENTICATE**
   Authentication required. relate to BTA_SEC_AUTHENTICATE in bta/bta_api.h

**ESP_SPP_SEC_ENCRYPT**
   Encryption required. relate to BTA_SEC_ENCRYPT in bta/bta_api.h

**ESP_SPP_SEC_MODE4_LEVEL4**
   Mode 4 level 4 service, i.e. incoming/outgoing MITM and P-256 encryption relate to BTA_SEC_MODE4_LEVEL4 in bta/bta_api.h

**ESP_SPP_SEC_MITM**
   Man-In-The_Middle protection relate to BTA_SEC_MITM in bta/bta_api.h

**ESP_SPP_SEC_IN_16_DIGITS**
   Min 16 digit for pin code relate to BTA_SEC_IN_16_DIGITS in bta/bta_api.h

**Type Definitions**

typedef uint16_t esp_spp_sec_t
typedef void (*esp_spp_cb_t)(esp_spp_cb_event_t event, esp_spp_cb_param_t *param)

SPP callback function type. When handle ESP_SPP_DATA_IND_EVT, it is strongly recommended to cache incoming data, and process them in other lower priority application task rather than in this callback directly.

**Param event**  Event type  
**Param param**  Point to callback parameter, currently is union type

### Enumerations

**enum esp_spp_status_t**

Values:

- enum \_ESP\_SPP\_SUCCESS  Successful operation.
  - enum \_ESP\_SPP\_FAILURE  Generic failure.
  - enum \_ESP\_SPP\_BUSY  Temporarily can not handle this request.
  - enum \_ESP\_SPP\_NO\_DATA  No data
  - enum \_ESP\_SPP\_NO\_RESOURCE  No more resource
  - enum \_ESP\_SPP\_NEED\_INIT  SPP module shall init first
    - enum \_ESP\_SPP\_NEED\_DEINIT  SPP module shall deinit first
  - enum \_ESP\_SPP\_NO\_CONNECTION  Connection may have been closed
  - enum \_ESP\_SPP\_NO\_SERVER  No SPP server

**enum esp_spp_role_t**

Values:

- enum \_ESP\_SPP\_ROLE\_MASTER  Role: master
  - enum \_ESP\_SPP\_ROLE\_SLAVE  Role: slave

**enum esp_spp_mode_t**

Values:
enumerator `ESP_SPP_MODE_CB`
    When data is coming, a callback will come with data

enumerator `ESP_SPP_MODE_VFS`
    Use VFS to write/read data

e num `esp_spp_cb_event_t`
    SPP callback function events.
    Values:

    enumerator `ESP_SPP_INIT_EVT`
        When SPP is initialized, the event comes

    enumerator `ESP_SPP_UNINIT_EVT`
        When SPP is deinitialized, the event comes

    enumerator `ESP_SPP_DISCOVERY_COMP_EVT`
        When SDP discovery complete, the event comes

    enumerator `ESP_SPP_OPEN_EVT`
        When SPP Client connection open, the event comes

    enumerator `ESP_SPP_CLOSE_EVT`
        When SPP connection closed, the event comes

    enumerator `ESP_SPP_START_EVT`
        When SPP server started, the event comes

    enumerator `ESP_SPP_CL_INIT_EVT`
        When SPP client initiated a connection, the event comes

    enumerator `ESP_SPP_DATA_IND_EVT`
        When SPP connection received data, the event comes, only for ESP_SPP_MODE_CB

    enumerator `ESP_SPP_CONG_EVT`
        When SPP connection congestion status changed, the event comes, only for ESP_SPP_MODE_CB

    enumerator `ESP_SPP_WRITE_EVT`
        When SPP write operation completes, the event comes, only for ESP_SPP_MODE_CB

    enumerator `ESP_SPP_SRV_OPEN_EVT`
        When SPP Server connection open, the event comes

    enumerator `ESP_SPP_SRV_STOP_EVT`
        When SPP server stopped, the event comes

**HFP DEFINES**

**API Reference**
Header File

- components/bt/host/bluedroid/api/include/api/esphf_defs.h

Enumerations

enum esp_hf_in_band_ring_state_t
  in-band ring tone state
  Values:

  enumerator ESP_HF_IN_BAND_RINGTONE_NOT_PROVIDED

  enumerator ESP_HF_IN_BAND_RINGTONE_PROVIDED

enum esp_hf_vr_state_t
  voice recognition state
  Values:

  enumerator ESP_HF_VR_STATE_DISABLED
    voice recognition disabled

  enumerator ESP_HF_VR_STATE_ENABLED
    voice recognition enabled

enum esp_hf_volume_control_target_t
  Bluetooth HFP audio volume control target.
  Values:

  enumerator ESP_HF_VOLUME_CONTROL_TARGET_SPK
    speaker

  enumerator ESP_HF_VOLUME_CONTROL_TARGET_MIC
    microphone

enum esp_hf_audio_state_t
  Bluetooth HFP audio connection status.
  Values:

  enumerator ESP_HF_AUDIO_STATE_DISCONNECTED
    audio connection released

  enumerator ESP_HF_AUDIO_STATE_CONNECTING
    audio connection has been initiated

  enumerator ESP_HF_AUDIO_STATE_CONNECTED
    audio connection is established

  enumerator ESP_HF_AUDIO_STATE_CONNECTED_MSBC
    mSBC audio connection is established
enum `esp_hf_volume_type_t`

Values:

enumerator `ESP_HF_VOLUME_TYPE_SPK`

enumerator `ESP_HF_VOLUME_TYPE_MIC`

enum `esp_hf_network_state_t`

+CIND network service availability status

Values:

enumerator `ESP_HF_NETWORK_STATE_NOT_AVAILABLE`

enumerator `ESP_HF_NETWORK_STATE_AVAILABLE`

enum `esp_hf_service_type_t`

+CIEV Service type

Values:

enumerator `ESP_HF_SERVICE_TYPE_HOME`

enumerator `ESP_HF_SERVICE_TYPE_ROAMING`

enum `esp_hf_call_status_t`

+CIND call status indicator values

Values:

enumerator `ESP_HF_CALL_STATUS_NO_CALLS`

no call in progress

enumerator `ESP_HF_CALL_STATUS_CALL_IN_PROGRESS`

call is present (active or held)

enum `esp_hf_call_setup_status_t`

+CIND call setup status indicator values

Values:

enumerator `ESP_HF_CALL_SETUP_STATUS_IDLE`

no call setup in progress

enumerator `ESP_HF_CALL_SETUP_STATUS_INCOMING`

incoming call setup in progress

enumerator `ESP_HF_CALL_SETUP_STATUS_OUTGOING_DIALING`

outgoing call setup in dialing state
enumerator **ESP_HF_CALL_SETUP_STATUS_OUTGOING_ALERTING**

outgoing call setup in alerting state

**enum esp_hf_roaming_status_t**

+CIND roaming status indicator values

*Values:*

enumerator **ESP_HF_ROAMING_STATUS_INACTIVE**

roaming is not active

enumerator **ESP_HF_ROAMING_STATUS_ACTIVE**

a roaming is active

**enum esp_hf_call_held_status_t**

+CIND call held indicator values

*Values:*

enumerator **ESP_HF_CALL_HELD_STATUS_NONE**

no calls held

enumerator **ESP_HF_CALL_HELD_STATUS_HELD_AND_ACTIVE**

both active and held call

enumerator **ESP_HF_CALL_HELD_STATUS_HELD**

call on hold, no active call

**enum esp_hf_current_call_status_t**

+CLCC status of the call

*Values:*

enumerator **ESP_HF_CURRENT_CALL_STATUS_ACTIVE**

active

enumerator **ESP_HF_CURRENT_CALL_STATUS_HELD**

held

enumerator **ESP_HF_CURRENT_CALL_STATUS_DIALING**

dialing (outgoing calls only)

enumerator **ESP_HF_CURRENT_CALL_STATUS_ALERTING**

alerting (outgoing calls only)

enumerator **ESP_HF_CURRENT_CALL_STATUS_INCOMING**

incoming (incoming calls only)

enumerator **ESP_HF_CURRENT_CALL_STATUS_WAITING**

waiting (incoming calls only)
enumerator `ESP_HF_CURRENT_CALL_STATUS_HELD_BY_RESP_HOLD`
call held by response and hold

enum `esp_hf_current_call_direction_t`
+CLCC direction of the call

Values:

enumerator `ESP_HF_CURRENT_CALL_DIRECTION_OUTGOING`
outgoing

enumerator `ESP_HF_CURRENT_CALL_DIRECTION_INCOMING`
incoming

enum `esp_hf_current_call_mpty_type_t`
+CLCC multi-party call flag

Values:

enumerator `ESP_HF_CURRENT_CALL_MPTY_TYPE_SINGLE`
not a member of a multi-party call

enumerator `ESP_HF_CURRENT_CALL_MPTY_TYPE_MULTI`
member of a multi-party call

enum `esp_hf_current_call_mode_t`
+CLCC call mode

Values:

enumerator `ESP_HF_CURRENT_CALL_MODE_VOICE`

enumerator `ESP_HF_CURRENT_CALL_MODE_DATA`

enumerator `ESP_HF_CURRENT_CALL_MODE_FAX`

enum `esp_hf_call_addr_type_t`
+CLCC address type

Values:

enumerator `ESP_HF_CALL_ADDR_TYPE_UNKNOWN`
unknown address type

enumerator `ESP_HF_CALL_ADDR_TYPE_INTERNATIONAL`
international address

enum `esp_hf_subscriber_service_type_t`
+CNUM service type of the phone number

Values:
Chapter 2. API

enumerator **ESP_HF_SUBSCRIBER_SERVICE_TYPE_UNKNOWN**
unknown

enumerator **ESP_HF_SUBSCRIBER_SERVICE_TYPE_VOICE**
voice service

enumerator **ESP_HF_SUBSCRIBER_SERVICE_TYPE_FAX**
fax service

enum **esp_hf_btrh_status_t**
+BTRH response and hold result code

Values:

enumerator **ESP_HF_BTRH_STATUS_HELD**
incoming call is put on hold in AG

enumerator **ESP_HF_BTRH_STATUS_ACCEPTED**
held incoming call is accepted in AG

enumerator **ESP_HF_BTRH_STATUS_REJECTED**
held incoming call is rejected in AG

enum **esp_hf_btrh_cmd_t**
AT+BTRH response and hold action code.

Values:

enumerator **ESP_HF_BTRH_CMD_HOLD**
put the incoming call on hold

enumerator **ESP_HF_BTRH_CMD_ACCEPT**
accept a held incoming call

enumerator **ESP_HF_BTRH_CMD_REJECT**
reject a held incoming call

enum **esp_hf_nrec_t**
Values:

enumerator **ESP_HF_NREC_STOP**

enumerator **ESP_HF_NREC_START**

enum **esp_hf_call_waiting_status_t**
+CCWA response status

Values:

enumerator **ESP_HF_CALL_WAITING_INACTIVE**
enumerator **ESP_HF_CALL_WAITING_ACTIVE**

eNum **esp_hf_wbs_config_t**

Values:

enumerator **ESP_HF_WBS_NONE**

enumerator **ESP_HF_WBS_NO**

enumerator **ESP_HF_WBS_YES**

eNum **esp_hf_connection_state_t**

Bluetooth HFP RFCOMM connection and service level connection status.

Values:

enumerator **ESP_HF_CONNECTION_STATE_DISCONNECTED**

RFCOMM data link channel released

enumerator **ESP_HF_CONNECTION_STATE_CONNECTING**

connecting remote device on the RFCOMM data link

enumerator **ESP_HF_CONNECTION_STATE_CONNECTED**

RFCOMM connection established

enumerator **ESP_HF_CONNECTION_STATE_SLC_CONNECTED**

service level connection established

enumerator **ESP_HF_CONNECTION_STATE_DISCONNECTING**

disconnecting with remote device on the RFCOMM data link

eNum **esp_hf_chld_type_t**

AT+CHLD command values.

Values:

enumerator **ESP_HF_CHLD_TYPE_REL**

<0>, Terminate all held or set UDUB ( "busy”) to a waiting call

enumerator **ESP_HF_CHLD_TYPE_REL_ACC**

<1>, Terminate all active calls and accepts a waiting/held call

enumerator **ESP_HF_CHLD_TYPE_HOLD_ACC**

<2>, Hold all active calls and accepts a waiting/held call

enumerator **ESP_HF_CHLD_TYPE_MERGE**

<3>, Add all held calls to a conference

enumerator **ESP_HF_CHLD_TYPE_MERGE_DETACH**

<4>, connect the two calls and disconnects the subscriber from both calls
enumerator **ESP_HF_CHLD_TYPE_REL_X**

<1x>, releases specified calls only

enumerator **ESP_HF_CHLD_TYPE_PRIV_X**

<2x>, request private consultation mode with specified call

**enum esp_hf_at_response_code_t**

*Values:*

enumerator **ESP_HF_AT_RESPONSE_CODE_OK**

acknowledges execution of a command line

enumerator **ESP_HF_AT_RESPONSE_CODE_ERR**

command not accepted

enumerator **ESP_HF_AT_RESPONSE_CODE_NO_CARRIER**

connection terminated

enumerator **ESP_HF_AT_RESPONSE_CODE_BUSY**

busy signal detected

enumerator **ESP_HF_AT_RESPONSE_CODE_NO_ANSWER**

connection completion timeout

enumerator **ESP_HF_AT_RESPONSE_CODE_DELAYED**

delayed

enumerator **ESP_HF_AT_RESPONSE_CODE_BLACKLISTED**

blacklisted

enumerator **ESP_HF_AT_RESPONSE_CODE_CME**

CME error

**enum esp_hf_at_response_t**

*Values:*

enumerator **ESP_HF_AT_RESPONSE_ERROR**

enumerator **ESP_HF_AT_RESPONSE_OK**

**enum esp_hf_cme_err_t**

Extended Audio Gateway Error Result Code Response.

*Values:*

enumerator **ESP_HF_CME_AG_FAILURE**

ag failure
enumerator ESP_HF_CME_NO_CONNECTION_TO_PHONE
   no connection to phone
enumerator ESP_HF_CME_OPERATION_NOT_ALLOWED
   operation not allowed
enumerator ESP_HF_CME_OPERATION_NOT_SUPPORTED
   operation not supported
enumerator ESP_HF_CME_PH_SIM_PIN_REQUIRED
   PH-SIM PIN Required
enumerator ESP_HF_CME_SIM_NOT_INSERTED
   SIM not inserted
enumerator ESP_HF_CME_SIM_PIN_REQUIRED
   SIM PIN required
enumerator ESP_HF_CME_SIM_PUK_REQUIRED
   SIM PUK required
enumerator ESP_HF_CME_SIM_FAILURE
   SIM failure
enumerator ESP_HF_CME_SIM_BUSY
   SIM busy
enumerator ESP_HF_CME_INCORRECT_PASSWORD
   incorrect password
enumerator ESP_HF_CME_SIM_PIN2_REQUIRED
   SIM PIN2 required
enumerator ESP_HF_CME_SIM_PUK2_REQUIRED
   SIM PUK2 required
enumerator ESP_HF_CME_MEMORY_FULL
   memory full
enumerator ESP_HF_CME_INVALID_INDEX
   invalid index
enumerator ESP_HF_CME_MEMORY_FAILURE
   memory failure
enumerator ESP_HF_CME_TEXT_STRING_TOO_LONG
   test string too long
enumerator **ESP_HF_CME_INVALID_CHARACTERS_IN_TEXT_STRING**
invalid characters in text string

enumerator **ESP_HF_CME_DIAL_STRING_TOO_LONG**
dial string too long

enumerator **ESP_HF_CME_INVALID_CHARACTERS_IN_DIAL_STRING**
invalid characters in dial string

enumerator **ESP_HF_CME_NO_NETWORK_SERVICE**
no network service

enumerator **ESP_HF_CME_NETWORK_TIMEOUT**
network timeout

enumerator **ESP_HF_CME_NETWORK_NOT_ALLOWED**
network not allowed &quot;emergency calls only&quot;

**HFP CLIENT API**

**API Reference**

**Header File**

- components/bt/host/bluedroid/api/include/api/esp_hf_client_api.h

**Functions**

```c
esp_err_t esp_hf_client_register_callback(esp_hf_client_cb_t callback)
```

Register application callback function to HFP client module. This function should be called only after esp_bluedroid_enable() completes successfully.

参数 callback [in] HFP client event callback function

返回

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

```c
esp_err_t esp_hf_client_init(void)
```

Initialize the bluetooth HFP client module. This function should be called after esp_bluedroid_enable() completes successfully.

返回

- ESP_OK: if the initialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_deinit(void)
```

De-initialize for HFP client module. This function should be called only after esp_bluedroid_enable() completes successfully.

返回

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others
### esp_err_t esp_hf_client_connect (esp_bd_addr_t remote_bda)

Establish a Service Level Connection to remote bluetooth HFP audio gateway(AG) device. This function must be called after esp_hf_client_init() and before esp_hf_client_deinit().

- **remote_bda** [in] remote bluetooth device address
- **ESP_OK:** connect request is sent to lower layer
- **ESP_INVALID_STATE:** if bluetooth stack is not yet enabled
- **ESP_FAIL:** others

### esp_err_t esp_hf_client_disconnect (esp_bd_addr_t remote_bda)

Disconnect from the remote HFP audio gateway. This function must be called after esp_hf_client_init() and before esp_hf_client_deinit().

- **remote_bda** [in] remote bluetooth device address
- **ESP_OK:** disconnect request is sent to lower layer
- **ESP_INVALID_STATE:** if bluetooth stack is not yet enabled
- **ESP_FAIL:** others

### esp_err_t esp_hf_client_connect_audio (esp_bd_addr_t remote_bda)

Create audio connection with remote HFP AG. As a precondition to use this API, Service Level Connection shall exist with AG.

- **remote_bda** [in] remote bluetooth device address
- **ESP_OK:** disconnect request is sent to lower layer
- **ESP_INVALID_STATE:** if bluetooth stack is not yet enabled
- **ESP_FAIL:** others

### esp_err_t esp_hf_client_disconnect_audio (esp_bd_addr_t remote_bda)

Release the established audio connection with remote HFP AG. As a precondition to use this API, Service Level Connection shall exist with AG.

- **remote_bda** [in] remote bluetooth device address
- **ESP_OK:** disconnect request is sent to lower layer
- **ESP_INVALID_STATE:** if bluetooth stack is not yet enabled
- **ESP_FAIL:** others

### esp_err_t esp_hf_client_start_voice_recognition (void)

Enable voice recognition in the AG. As a precondition to use this API, Service Level Connection shall exist with AG.

- **ESP_OK:** disconnect request is sent to lower layer
- **ESP_INVALID_STATE:** if bluetooth stack is not yet enabled
- **ESP_FAIL:** others

### esp_err_t esp_hf_client_stop_voice_recognition (void)

Disable voice recognition in the AG. As a precondition to use this API, Service Level Connection shall exist with AG.

- **ESP_OK:** disconnect request is sent to lower layer
- **ESP_INVALID_STATE:** if bluetooth stack is not yet enabled
- **ESP_FAIL:** others

### esp_err_t esp_hf_client_volume_update (esp_hf_volume_control_target_t type, int volume)

Volume synchronization with AG. As a precondition to use this API, Service Level Connection shall exist with AG.

- **type**
- **volume**
Chapter 2. API

- **type** [in] volume control target, speaker or microphone
- **volume** [in] gain of the speaker of microphone, ranges 0 to 15

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_dial (const char *number)
```

Place a call with a specified number, if number is NULL, last called number is called. As a precondition to use this API, Service Level Connection shall exist with AG.

参数 **number** [in] number string of the call. If NULL, the last number is called (aka re-dial)

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_dial_memory (int location)
```

Place a call with number specified by location(speed dial). As a precondition to use this API, Service Level Connection shall exist with AG.

参数 **location** [in] location of the number in the memory

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_send_chld_cmd (esp_hf_chld_type_t chld, int idx)
```

Send call hold and multiparty commands, or enhanced call control commands(Use AT+CHLD). As a precondition to use this API, Service Level Connection shall exist with AG.

参数
- **chld** [in] AT+CHLD call hold and multiparty handling AT command.
- **idx** [in] used in Enhanced Call Control Mechanisms, used if chld is ESP_HF_CHLD_TYPE_REL_X or ESP_HF_CHLD_TYPE_PRIV_X

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_send_btrh_cmd (esp_hf_btrh_cmd_t btrh)
```

Send response and hold action command(Send AT+BTRH command) As a precondition to use this API, Service Level Connection shall exist with AG.

参数 **btrh** [in] response and hold action to send

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_answer_call (void)
```

Answer an incoming call(send ATA command). As a precondition to use this API, Service Level Connection shall exist with AG.

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_reject_call (void)
```

Reject an incoming call(send AT+CHUP command). As a precondition to use this API, Service Level Connection shall exist with AG.
Section 2. API Reference

**esp_err_t esp_hf_client_query_current_calls (void)**
Query list of current calls in AG (send AT+CLCC command). As a precondition to use this API, Service Level Connection shall exist with AG.

**esp_err_t esp_hf_client_query_current_operator_name (void)**
Query the name of currently selected network operator in AG (use AT+COPS commands). As a precondition to use this API, Service Level Connection shall exist with AG.

**esp_err_t esp_hf_client_retrieve_subscriber_info (void)**
Get subscriber information number from AG (send AT+CNUM command). As a precondition to use this API, Service Level Connection shall exist with AG.

**esp_err_t esp_hf_client_send_dtmf (char code)**
Transmit DTMF codes during an ongoing call (use AT+VTS commands). As a precondition to use this API, Service Level Connection shall exist with AG.

**esp_err_t esp_hf_client_send_xapl (char* information, uint32_t features)**
Send command to enable Vendor specific feature to indicate battery level and docker status. This is Apple-specific commands, but used by most device, including Android and Windows.

**esp_err_t esp_hf_client_send_iphoneaccev (uint32_t bat_level, bool docked)**
Send Battery level and docker status Enable this feature using XAPL command first. This is Apple-specific commands, but used by most device, including Android and Windows.
Chapter 2. API

- **bat_level** - [in] Battery Level: value between 0 and 9
- **docked** - [in] Dock State: false = undocked, true = docked

返回
- ESP_OK: battery level is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_request_last_voice_tag_number (void)
```

Request a phone number from AG corresponding to last voice tag recorded (send AT+BINP command). As a precondition to use this API, Service Level Connection shall exist with AG.

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_send_nrec (void)
```

Disable echo cancellation and noise reduction in the AG (use AT+NREC=0 command). As a precondition to use this API, Service Level Connection shall exist with AG.

返回
- ESP_OK: NREC=0 request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_client_register_data_callback (esp_hf_client_incoming_data_cb_t recv,
                                               esp_hf_client_outgoing_data_cb_t send)
```

Register HFP client data output function; the callback is only used in the case that Voice Over HCI is enabled.

参数
- **recv** - [in] HFP client incoming data callback function
- **send** - [in] HFP client outgoing data callback function

返回
- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

```c
void esp_hf_client_outgoing_data_ready (void)
```

Trigger the lower-layer to fetch and send audio data. This function is only only used in the case that Voice Over HCI is enabled. After this function is called, lower layer will invoke esp_hf_client_outgoing_data_cb_t to fetch data.

As a precondition to use this API, Service Level Connection shall exist with AG.

```c
void esp_hf_client_pcm_resample_deinit (void)
```

Deinitialize the down sampling converter.

```c
void esp_hf_client_pcm_resample (uint32_t src_sps, uint32_t bits, uint32_t channels)
```

Initialize the down sampling converter. This is a utility function that can only be used in the case that Voice Over HCI is enabled.

参数
- **src_sps** - [in] original samples per second(source audio data, i.e. 48000, 32000, 16000, 44100, 22050, 11025)
- **bits** - [in] number of bits per pcm sample (16)
- **channels** - [in] number of channels (i.e. mono(1), stereo(2)…)

```c
esp_err_t esp_hf_client_pcm_resample (void *src, uint32_t in_bytes, void *dst)
```

Down sampling utility to convert high sampling rate into 8K/16bits 1-channel mode PCM samples. This can only be used in the case that Voice Over HCI is enabled.

参数
- **src** - [in] pointer to the buffer where the original sampling PCM are stored
Chapter 2. API

- **in_bytes** - [in] length of the input PCM sample buffer in byte
- **dst** - [in] pointer to the buffer which is to be used to store the converted PCM samples
  
  number of samples converted

Unions

union esp_hf_client_cb_param_t

```
#include <esp_hf_client_api.h>  // HFP client callback parameters.
```

Public Members

```
struct esp_hf_client_cb_param_t::hf_client_conn_stat_param conn_stat
  // HF callback param of ESP_HF_CLIENT_CONNECTION_STATE_EVT

struct esp_hf_client_cb_param_t::hf_client_audio_stat_param audio_stat
  // HF callback param of ESP_HF_CLIENT_AUDIO_STATE_EVT

struct esp_hf_client_cb_param_t::hf_client_bvra_param bvra
  // HF callback param of ESP_HF_CLIENT_BVRA_EVT

struct esp_hf_client_cb_param_t::hf_client_service_availability_param service_availability
  // HF callback param of ESP_HF_CLIENT_CIND_SERVICE_AVAILABILITY_EVT

struct esp_hf_client_cb_param_t::hf_client_network_roaming_param roaming
  // HF callback param of ESP_HF_CLIENT_CIND_ROAMING_STATUS_EVT

struct esp_hf_client_cb_param_t::hf_client_signal_strength_ind_param signal_strength
  // HF callback param of ESP_HF_CLIENT_CIND_SIGNAL_STRENGTH_EVT

struct esp_hf_client_cb_param_t::hf_client_battery_level_ind_param battery_level
  // HF callback param of ESP_HF_CLIENT_CIND_BATTERY_LEVEL_EVT

struct esp_hf_client_cb_param_t::hf_client_current_operator_param cops
  // HF callback param of ESP_HF_CLIENT_COPS_CURRENT_OPERATOR_EVT

struct esp_hf_client_cb_param_t::hf_client_call_ind_param call
  // HF callback param of ESP_HF_CLIENT_CIND_CALL_EVT

struct esp_hf_client_cb_param_t::hf_client_call_setup_ind_param call_setup
  // HF callback param of ESP_HF_CLIENT_BVRA_EVT

struct esp_hf_client_cb_param_t::hf_client_call_held_ind_param call_held
  // HF callback param of ESP_HF_CLIENT_CIND_CALL_HELD_EVT

struct esp_hf_client_cb_param_t::hf_client_btrh_param btrh
  // HF callback param of ESP_HF_CLIENT_BRTH_EVT

struct esp_hf_client_cb_param_t::hf_client_clip_param clip
  // HF callback param of ESP_HF_CLIENT_CLIP_EVT
```
struct esp_hf_client_cb_param_t::hf_client_ccwa_param ccwa
    HF callback param of ESP_HF_CLIENT_BVRA_EVT

struct esp_hf_client_cb_param_t::hf_client_clcc_param clcc
    HF callback param of ESP_HF_CLIENT_CLCC_EVT

struct esp_hf_client_cb_param_t::hf_client_volume_control_param volume_control
    HF callback param of ESP_HF_CLIENT_VOLUME_CONTROL_EVT

struct esp_hf_client_cb_param_t::hf_client_at_response_param at_response
    HF callback param of ESP_HF_CLIENT_AT_RESPONSE_EVT

struct esp_hf_client_cb_param_t::hf_client_cnum_param cnum
    HF callback param of ESP_HF_CLIENT_CNUM_EVT

struct esp_hf_client_cb_param_t::hf_client_bsirparam bsir
    HF callback param of ESP_HF_CLIENT_BSIR_EVT

struct esp_hf_client_cb_param_t::hf_client_binp_param binp
    HF callback param of ESP_HF_CLIENT_BINP_EVT

struct hf_client_at_response_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_AT_RESPONSE_EVT.

Public Members

    esp_hf_at_response_code_t code
        AT response code

    esp_hf_cme_err_t cme
        Extended Audio Gateway Error Result Code

struct hf_client_audio_stat_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_AUDIO_STATE_EVT.

Public Members

    esp_hf_client_audio_state_t state
        audio connection state

    esp_bd_addr_t remote_bda
        remote bluetooth device address

struct hf_client_battery_level_ind_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_BATTERY_LEVEL_EVT.
Chapter 2. API 参考

**Public Members**

```c
int value  
// battery charge value, ranges from 0 to 5
```

```c
#include <esp_hf_client_api.h> ESP_HF_CLIENT_BINP_EVT.
```

**Public Members**

```c
const char *number  
// phone number corresponding to the last voice tag in the HF
```

```c
#include <esp_hf_client_api.h> ESP_HF_CLIENT BSIR_EVT.
```

**Public Members**

```c
esp_hf_client_in_band_ring_state_t state  
// setting state of in-band ring tone
```

```c
#include <esp_hf_client_api.h> ESP_HF_CLIENT_BTRH_EVT.
```

**Public Members**

```c
esp_hf_btrh_status_t status  
// call hold and response status result code
```

```c
#include <esp_hf_client_api.h> ESP_HF_CLIENT BVRA_EVT.
```

**Public Members**

```c
esp_hf_vr_state_t value  
// voice recognition state
```

```c
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_CALL_HELD_EVT.
```

**Public Members**

```c
esp_hf_call_held_status_t status  
// bluetooth proprietary call hold status indicator
```
struct hf_client_call_ind_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_CALL_EVT.

Public Members

esp_hf_call_status_t status
call status indicator

struct hf_client_call_setup_ind_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_CALL_SETUP_EVT.

Public Members

esp_hf_call_setup_status_t status
call setup status indicator

struct hf_client_ccwa_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CCWA_EVT.

Public Members

const char *number
phone number string of waiting call

struct hf_client_clcc_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CLCC_EVT.

Public Members

int idx
numbering (starting with 1) of the call

esp_hf_current_call_direction_t dir
direction of the call

esp_hf_current_call_status_t status
status of the call

esp_hf_current_call_mpty_type_t mpty
multi-party flag

char *number
phone number (optional)

struct hf_client_clip_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CLIP_EVT.
Public Members

```
const char *number
    phone number string of call
```

```c
struct hf_client_cnum_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_CNUM_EVT.
```

Public Members

```
const char *number
    phone number string
```

```c
esp_hf_subscriber_service_type_t type
    service type that the phone number relates to
```

```c
struct hf_client_conn_stat_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_CONNECTION_STATE_EVT.
```

Public Members

```
esp_hf_client_connection_state_t state
    HF connection state
```

```c
uint32_t peer_feat
    AG supported features
```

```c
uint32_t child_feat
    AG supported features on call hold and multiparty services
```

```c
esp_bd_addr_t remote_bda
    remote bluetooth device address
```

```c
struct hf_client_current_operator_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_COPS_CURRENT_OPERATOR_EVT.
```

Public Members

```
const char *name
    name of the network operator
```

```c
struct hf_client_network_roaming_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_ROAMING_STATUS_EVT.
```
**Public Members**

```c
esp_hf_roaming_status_t status
```
roaming status

```c
struct hf_client_service_availability_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_SERVICE_AVAILABILITY_EVT.
```

**Public Members**

```c
esp_hf_network_state_t status
```
service availability status

```c
struct hf_client_signal_strength_ind_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_SIGNAL_STRENGTH_EVT.
```

**Public Members**

```c
int value
```
signal strength value, ranges from 0 to 5

```c
struct hf_client_volume_control_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_VOLUME_CONTROL_EVT.
```

**Public Members**

```c
esp_hf_volume_control_target_t type
```
volume control target, speaker or microphone

```c
int volume
```
gain, ranges from 0 to 15

**Macros**

```c
ESP_BT_HF_CLIENT_NUMBER_LEN
ESP_BT_HF_CLIENT_OPERATOR_NAME_LEN
ESP_BT_HF_AT_SEND_XAPL_LEN
ESP_HF_CLIENT_PEER_FEAT_3WAY
ESP_HF_CLIENT_PEER_FEAT_ECNR
ESP_HF_CLIENT_PEER_FEAT_VREC
```
ESP_HF_CLIENT_PEER_FEAT_INBAND
ESP_HF_CLIENT_PEER_FEAT_VTAG
ESP_HF_CLIENT_PEER_FEAT_REJECT
ESP_HF_CLIENT_PEER_FEAT_ECS
ESP_HF_CLIENT_PEER_FEAT_ECC
ESP_HF_CLIENT_PEER_FEAT_EXERR
ESP_HF_CLIENT_PEER_FEAT_CODEC
ESP_HF_CLIENT_PEER_FEAT_HF_IND
ESP_HF_CLIENT_PEER_FEAT_ESCO_S4
ESP_HF_CLIENT_CHLD_FEAT_REL
ESP_HF_CLIENT_CHLD_FEAT_REL_ACC
ESP_HF_CLIENT_CHLD_FEAT_REL_X
ESP_HF_CLIENT_CHLD_FEAT_HOLD_ACC
ESP_HF_CLIENT_CHLD_FEAT_PRIV_X
ESP_HF_CLIENT_CHLD_FEAT_MERGE
ESP_HF_CLIENT_CHLD_FEAT_MERGE_DETACH
ESP_HF_CLIENT_XAPL_FEAT_RESERVED
ESP_HF_CLIENT_XAPL_FEAT_BATTERY_REPORT
ESP_HF_CLIENT_XAPL_FEAT_DOCKED
ESP_HF_CLIENT_XAPL_FEAT_SIRI_STATUS_REPORT
ESP_HF_CLIENT_XAPL_NR_STATUS_REPORT
**Type Definitions**

typedef void (*esp_hf_client_incoming_data_cb_t)(const uint8_t *buf, uint32_t len)

HFP client incoming data callback function, the callback is useful in case of Voice Over HCI.

**Param buf** [in]: pointer to incoming data (payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.

**Param len** [in]: size (in bytes) in buf

typedef uint32_t (*esp_hf_client_outgoing_data_cb_t)(uint8_t *buf, uint32_t len)

HFP client outgoing data callback function, the callback is useful in case of Voice Over HCI. Once audio connection is set up and the application layer has prepared data to send, the lower layer will call this function to read data and then send. This callback is supposed to be implemented as non-blocking, and if data is not enough, return value 0 is supposed.

**Param buf** [in]: pointer to incoming data (payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.

**Param len** [in]: size (in bytes) in buf

**Return** length of data successfully read

typedef void (*esp_hf_client_cb_t)(esp_hf_client_cb_event_t event, esp_hf_client_cb_param_t *param)

HFP client callback function type.

**Param event**: Event type

**Param param**: Pointer to callback parameter

**Enumerations**

enum esp_hf_client_connection_state_t

Bluetooth HFP RFCOMM connection and service level connection status.

**Values**:

enumerator ESP_HF_CLIENT_CONNECTION_STATE_DISCONNECTED

RFCOMM data link channel released

enumerator ESP_HF_CLIENT_CONNECTION_STATE_CONNECTING

connecting remote device on the RFCOMM data link

enumerator ESP_HF_CLIENT_CONNECTION_STATE_CONNECTED

RFCOMM connection established

enumerator ESP_HF_CLIENT_CONNECTION_STATE_SLC_CONNECTED

service level connection established

enumerator ESP_HF_CLIENT_CONNECTION_STATE_DISCONNECTING

disconnecting with remote device on the RFCOMM dat link

enum esp_hf_client_audio_state_t

Bluetooth HFP audio connection status.

**Values**:
Chapter 2. API Reference

**enumerator** `ESP_HF_CLIENT_AUDIO_STATE_DISCONNECTED`
audio connection released

**enumerator** `ESP_HF_CLIENT_AUDIO_STATE_CONNECTING`
audio connection has been initiated

**enumerator** `ESP_HF_CLIENT_AUDIO_STATE_CONNECTED`
audio connection is established

**enumerator** `ESP_HF_CLIENT_AUDIO_STATE_CONNECTED_MSBC`
mSBC audio connection is established

**enum** `esp_hf_client_in_band_ring_state_t`
in-band ring tone state

*Values:*

**enumerator** `ESP_HF_CLIENT_IN_BAND_RINGTONE_NOT_PROVIDED`

**enumerator** `ESP_HF_CLIENT_IN_BAND_RINGTONE_PROVIDED`

**enum** `esp_hf_client_cb_event_t`
HF CLIENT callback events.

*Values:*

**enumerator** `ESP_HF_CLIENT_CONNECTION_STATE_EVT`
connection state changed event

**enumerator** `ESP_HF_CLIENT_AUDIO_STATE_EVT`
audio connection state change event

**enumerator** `ESP_HF_CLIENT_BVRA_EVT`
voice recognition state change event

**enumerator** `ESP_HF_CLIENT_CIND_CALL_EVT`
call indication

**enumerator** `ESP_HF_CLIENT_CIND_CALL_SETUP_EVT`
call setup indication

**enumerator** `ESP_HF_CLIENT_CIND_CALL_HELD_EVT`
call held indication

**enumerator** `ESP_HF_CLIENT_CIND_SERVICE_AVAILABILITY_EVT`
network service availability indication

**enumerator** `ESP_HF_CLIENT_CIND_SIGNAL_STRENGTH_EVT`
signal strength indication
enumerator **ESP_HF_CLIENT_CIND_ROAMING_STATUS_EVT**  
roaming status indication

enumerator **ESP_HF_CLIENT_CIND_BATTERY_LEVEL_EVT**  
battery level indication

enumerator **ESP_HF_CLIENT_COPS_CURRENT_OPERATOR_EVT**  
current operator information

enumerator **ESP_HF_CLIENT_BTRH_EVT**  
call response and hold event

enumerator **ESP_HF_CLIENT_CLIP_EVT**  
Calling Line Identification notification

enumerator **ESP_HF_CLIENT_CCWA_EVT**  
call waiting notification

enumerator **ESP_HF_CLIENT_CLCC_EVT**  
list of current calls notification

enumerator **ESP_HF_CLIENT_VOLUME_CONTROL_EVT**  
audio volume control command from AG, provided by +VGM or +VGS message

enumerator **ESP_HF_CLIENT_AT_RESPONSE_EVT**  
AT command response event

enumerator **ESP_HF_CLIENT_CNUM_EVT**  
subscriber information response from AG

enumerator **ESP_HF_CLIENT_BSIR_EVT**  
setting of in-band ring tone

enumerator **ESP_HF_CLIENT_BINP_EVT**  
requested number of last voice tag from AG

enumerator **ESP_HF_CLIENT_RING_IND_EVT**  
ring indication event

### HFP AG API

### API Reference

**Header File**

- components/bt/host/bluedroid/api/include/api/esp_hf_ag_api.h
### Functions

**esp_err_t esp_hf_ag_register_callback(esp_hf_cb_t callback)**

Register application callback function to HFP AG module. This function should be called only after esp_bluedroid_enable() completes successfully.

参数 `callback` - [in] HFP AG event callback function

返回

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

**esp_err_t esp_hf_ag_init (void)**

Initialize the bluetooth HF AG module. This function should be called after esp_bluedroid_enable() completes successfully.

返回

- ESP_OK: if the initialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_deinit (void)**

De-initialize for HF AG module. This function should be called only after esp_bluedroid_enable() completes successfully.

返回

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_slc_connect (esp_bd_addr_t remote_bda)**

To establish a Service Level Connection to remote bluetooth HFP client device. This function must be called after esp_hf_ag_init() and before esp_hf_ag_deinit().

参数 `remote_bda` - [in] remote bluetooth HFP client device address

返回

- ESP_OK: connect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_slc_disconnect (esp_bd_addr_t remote_bda)**

Disconnect from the remote HFP client. This function must be called after esp_hf_ag_init() and before esp_hf_ag_deinit().

参数 `remote_bda` - [in] remote bluetooth device address

返回

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_audio_connect (esp_bd_addr_t remote_bda)**

Create audio connection with remote HFP client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数 `remote_bda` - [in] remote bluetooth device address

返回

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_audio_disconnect (esp_bd_addr_t remote_bda)**

Release the established audio connection with remote HFP client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数 `remote_bda` - [in] remote bluetooth device address
### Chapter 2. API 参考

**esp_err_t esp_hf_ag_vra_control(esp_bd_addr_t remote_bda, esp_hf_vr_state_t value)**

Response of Volume Recognition Command (AT+VRA) from HFP client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

**参数**
- `remote_bda` [in] the device address of voice recognition initiator
- `value` [in] 0 - voice recognition disabled, 1 - voice recognition enabled

**返回**
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if Bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_volume_control(esp_bd_addr_t remote_bda, esp_hf_volume_control_target_t type, int volume)**

Volume synchronization with HFP client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

**参数**
- `remote_bda` [in] remote Bluetooth device address
- `type` [in] volume control target, speaker or microphone
- `volume` [in] gain of the speaker or microphone, ranges 0 to 15

**返回**
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if Bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_unknown_at_send(esp_bd_addr_t remote_addr, char *unat)**

Handle Unknown AT command from HFP Client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

**参数**
- `remote_addr` [in] remote Bluetooth device address
- `unat` [in] User AT command response to HF Client. It will response “ERROR” by default if unat is NULL.

**返回**
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if Bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_cmee_send(esp_bd_addr_t remote_bda, esp_hf_at_response_code_t response_code, esp_hf_cme_err_t error_code)**

Unsolicited send extend AT error code to HFP Client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

**参数**
- `remote_bda` [in] remote Bluetooth device address
- `response_code` [in] AT command response code
- `error_code` [in] CME error code

**返回**
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if Bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_devices_status_indchange(esp_bd_addr_t remote_addr, esp_hf_call_status_t call_state, esp_hf_call_setup_status_t call_setup_state, esp_hf_network_state_t ntk_state, int signal)**
Unsolicited send device status notification to HFP Client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数
- remote_addr - [in] remote bluetooth device address
- call_state - [in] call state
- call_setup_state - [in] call setup state
- ntk_state - [in] network service state
- signal - [in] signal strength from 0 to 5

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_ag_cind_response(esp_bd_addr_t remote_addr, esp_hf_call_status_t call_state, esp_hf_call_setup_status_t call_setup_state, esp_hf_network_state_t ntk_state, int signal, esp_hf_roaming_status_t roam, int batt_lev, esp_hf_call_held_status_t call_held_status)
```

Response to device individual indicators to HFP Client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数
- remote_addr - [in] remote bluetooth device address
- call_state - [in] call state
- call_setup_state - [in] call setup state
- ntk_state - [in] network service state
- signal - [in] signal strength from 0 to 5
- roam - [in] roam state
- batt_lev - [in] battery level from 0 to 5
- call_held_status - [in] call held status

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_ag_cops_response(esp_bd_addr_t remote_addr, char *name)
```

Response for AT+COPS command from HF Client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数
- remote_addr - [in] remote bluetooth device address
- name - [in] current operator name

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

```c
esp_err_t esp_hf_ag_clcc_response(esp_bd_addr_t remote_addr, int index, esp_hf_current_call_direction_t dir, esp_hf_current_call_status_t current_call_state, esp_hf_current_call_mode_t mode, esp_hf_current_call_empty_type_t mpty, char *number, esp_hf_call_addr_type_t type)
```

Response to AT+CLCC command from HFP Client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数
- remote_addr - [in] remote bluetooth device address
- index - [in] the index of current call
- dir - [in] call direction (incoming/outgoing)
- current_call_state - [in] current call state
Chapter 2. API

**esp_err_t esp_hf_ag_cnum_response (esp_bd_addr_t remote_addr, char* number, esp hf subscriber service type_t type)**

Response for AT+CNUM command from HF Client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

**Parameters**
- **remote_addr** - [in] remote bluetooth device address
- **number** - [in] registration number
- **type** - [in] service type (unknown/voice/fax)

**Return**
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_bsir (esp_bd_addr_t remote_addr, esp hf in band ring state_t state)**

Inform HF Client that AG Provided in-band ring tone or not. As a precondition to use this API, Service Level Connection shall exist with HFP client.

**Parameters**
- **remote_addr** - [in] remote bluetooth device address
- **state** - [in] in-band ring tone state

**Return**
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_answer_call (esp_bd_addr_t remote_addr, int num_active, int num_held, esp hf call status_t call state, esp hf call setup status_t call setup state, char* number, esp hf call addr type_t call addr type)**

Answer Incoming Call from AG. As a precondition to use this API, Service Level Connection shall exist with HFP client.

**Parameters**
- **remote_addr** - [in] remote bluetooth device address
- **num_active** - [in] the number of active call
- **num_held** - [in] the number of held call
- **call_state** - [in] call state
- **call_setup_state** - [in] call setup state
- **number** - [in] number of the incoming call
- **call_addr_type** - [in] call address type

**Return**
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_reject_call (esp_bd_addr_t remote_addr, int num_active, int num_held, esp hf call status_t call state, esp hf call setup status_t call setup state, char* number, esp hf call addr type_t call addr_type)**

Reject Incoming Call from AG. As a precondition to use this API, Service Level Connection shall exist with HFP client.
Chapter 2. API 参考

参数
- remote_addr [in] remote bluetooth device address
- num_active [in] the number of active call
- num_held [in] the number of held call
- call_state [in] call state
- call_setup_state [in] call setup state
- number [in] number of the incoming call
- call_addr_type [in] call address type

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_out_call**

```c
esp_hf_ag_out_call(esp_bd_addr_t remote_addr, int num_active, int num_held,
                    esp_hf_call_status_t call_state, esp_hf_call_setup_status_t call_setup_state,
                    char *number, esp_hf_call_addr_type_t call_addr_type)
```

Initiate a call from AG. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数
- remote_addr [in] remote bluetooth device address
- num_active [in] the number of active call
- num_held [in] the number of held call
- call_state [in] call state
- call_setup_state [in] call setup state
- number [in] number of the outgoing call
- call_addr_type [in] call address type

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_end_call**

```c
esp_hf_ag_end_call(esp_bd_addr_t remote_addr, int num_active, int num_held,
                   esp_hf_call_status_t call_state, esp_hf_call_setup_status_t call_setup_state,
                   char *number, esp_hf_call_addr_type_t call_addr_type)
```

End an ongoing call. As a precondition to use this API, Service Level Connection shall exist with HFP client.

参数
- remote_addr [in] remote bluetooth device address
- num_active [in] the number of active call
- num_held [in] the number of held call
- call_state [in] call state
- call_setup_state [in] call setup state
- number [in] number of the call
- call_addr_type [in] call address type

返回
- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

**esp_err_t esp_hf_ag_register_data_callback**

```c
esp_hf_ag_register_data_callback(esp_hf_incoming_data_cb_t recv,
                                 esp_hf_outgoing_data_cb_t send)
```

Register AG data output function. The callback is only used in the case that Voice Over HCI is enabled.

参数
- recv [in] HFP client incoming data callback function
- send [in] HFP client outgoing data callback function

返回
- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer
Chapter 2. API 参考

void **esp_hf_ag_outgoing_data_ready** (void)

Trigger the lower-layertofetch and send audio data.

This function is only used in the case that Voice Over HCI is enabled.

As a precondition to use this API, Service Level Connection shall exist with HFP client.

After this function is called, lower layer will invoke esp_hf_client_outgoing_data_cb_t to fetch data

Unions

union **esp_hf_cb_param_t**

#include <esp_hf_ag_api.h> HFP AG callback parameters.

Public Members

struct **esp_hf_cb_param_t::hf_conn_stat_param** conn_stat

AG callback param of ESP_HF_CONNECTION_STATE_EVT

struct **esp_hf_cb_param_t::hf_audio_stat_param** audio_stat

AG callback param of ESP_HF_AUDIO_STATE_EVT

struct **esp_hf_cb_param_t::hf_vra_rep_param** vra_rep

AG callback param of ESP_HF_BVRA_RESPONSE_EVT

struct **esp_hf_cb_param_t::hf_volume_control_param** volume_control

AG callback param of ESP_HF_VOLUME_CONTROL_EVT

struct **esp_hf_cb_param_t::hf_unat_rep_param** unat_rep

AG callback param of ESP_HF_UNAT_RESPONSE_EVT

struct **esp_hf_cb_param_t::hf_cind_param** cind

AG callback param of ESP_HF_CIND_RESPONSE_EVT

struct **esp_hf_cb_param_t::hf_out_call_param** out_call

AG callback param of ESP_HF_DIAL_EVT

struct **esp_hf_cb_param_t::hf_vts_rep_param** vts_rep

AG callback param of ESP_HF_VTS_RESPONSE_EVT

struct **esp_hf_cb_param_t::hf_nrec_param** nrec

AG callback param of ESP_HF_NREC_RESPONSE_EVT

struct **esp_hf_cb_param_t::hf_wbs_rep_param** wbs_rep

AG callback param of ESP_HF_WBS_RESPONSE_EVT

struct **esp_hf_cb_param_t::hf_bcs_rep_param** bcs_rep

AG callback param of ESP_HF_BCS_RESPONSE_EVT
struct hf_audio_stat_param
#include <esp_hf_ag_api.h> ESP_HF_AUDIO_STATE_EVT.

Public Members

  esp_bd_addr_t remote_addr
  Remote bluetooth device address

  esp_hf_audio_state_t state
  Audio connection state

struct hf_bcs_rep_param
#include <esp_hf_ag_api.h> ESP_HF_BCS_RESPONSE_EVT.

Public Members

  esp_hf_wbs_config_t mode
  codec mode CVSD or mSBC

struct hf_cind_param
#include <esp_hf_ag_api.h> ESP_HF_CIND_RESPONSE_EVT.

Public Members

  esp_hf_call_status_t call_status
  call status indicator

  esp_hf_call_setup_status_t call_setup_status
  call setup status indicator

  esp_hf_network_state_t svc
  bluetooth proprietary call hold status indicator

  int signal_strength
  bluetooth proprietary call hold status indicator

  esp_hf_roaming_status_t roam
  bluetooth proprietary call hold status indicator

  int battery_level
  battery charge value, ranges from 0 to 5

  esp_hf_call_held_status_t call_held_status
  bluetooth proprietary call hold status indicator

struct hf_conn_stat_param
#include <esp_hf_ag_api.h> ESP_HS_CONNECTION_STATE_EVT.
Public Members

`esp_bd_addr_t remote_bda`
Remote bluetooth device address

`esp_hf_connection_state_t state`
Connection state

`uint32_t peer_feat`
HF supported features

`uint32_t chld_feat`
AG supported features on call hold and multiparty services

```c
#include <esp_hf_ag_api.h>
ESP_HF_NREC_RESPONSE_EVT.
```

Public Members

`esp_hf_nrec_t state`
NREC enabled or disabled

```c
#include <esp_hf_ag_api.h>
ESP_HF_DIAL_EVT.
```

Public Members

`esp_bd_addr_t remote_addr`
remote bluetooth device address

```c
char *num_or_loc
```
location in phone memory

```c
#include <esp_hf_ag_api.h>
ESP_HF_UNAT_RESPONSE_EVT.
```

Public Members

```c
char *unat
```
Unknown AT command string

```c
#include <esp_hf_ag_api.h>
ESP_HF_VOLUME_CONTROL_EVT.
```
Public Members

```c
struct hf_vra_rep_param
#include <esp_hf_ag_api.h> ESP_HF_BVRA_RESPONSE_EVT.
```

Public Members

```c
esp_bd_addr_t remote_addr
Remote bluetooth device address
```

```c
esp_hf_vr_state_t value
Voice recognition state
```

Public Members

```c
char * code
MTF code from HF Client
```

Public Members

```c
esp_hf_wbs_config_t codec
codec mode CVSD or mSBC
```

Macros

```c
ESP_HF_PEER_FEAT_3WAY
ESP_HF_PEER_FEAT_ECNR
ESP_HF_PEER_FEAT_VREC
ESP_HF_PEER_FEAT_INBAND
ESP_HF_PEER_FEAT_VTAG
```
ESP_HF_PEER_FEAT_REJECT
ESP_HF_PEER_FEAT_ECS
ESP_HF_PEER_FEAT_ECC
ESP_HF_PEER_FEAT_EXTERRE
ESP_HF_PEER_FEAT_CODEC
ESP_HF_PEER_FEAT_HF_IND
ESP_HF_PEER_FEAT_ESCO_S4
ESP_HF_CHLD_FEAT_REL
ESP_HF_CHLD_FEAT_REL_ACC
ESP_HF_CHLD_FEAT_REL_X
ESP_HF_CHLD_FEAT_HOLD_ACC
ESP_HF_CHLD_FEAT_PRIV_X
ESP_HF_CHLD_FEAT_MERGE
ESP_HF_CHLD_FEAT_MERGE_DETACH

### Type Definitions

typedef void (*esp_hf_incoming_data_cb_t)(const uint8_t *buf, uint32_t len)

AG incoming data callback function, the callback is useful in case of Voice Over HCI.

- **Param buf [in]**: pointer to incoming data (payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.
- **Param len [in]**: size (in bytes) in buf

typedef uint32_t (*esp_hf_outgoing_data_cb_t)(uint8_t *buf, uint32_t len)

AG outgoing data callback function, the callback is useful in case of Voice Over HCI. Once audio connection is set up and the application layer has prepared data to send, the lower layer will call this function to read data and then send. This callback is supposed to be implemented as non-blocking, and if data is not enough, return value 0 is supposed.

- **Param buf [in]**: pointer to incoming data (payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.
- **Param len [in]**: size (in bytes) in buf
- **Return** length of data successfully read
typedef void (*esp_hf_cb_t)(esp_hf_cb_event_t event, esp_hf_cb_param_t *param)

HF AG callback function type.

**Param event**: Event type
**Param param**: Pointer to callback parameter

**Enumerations**

enum esp_hf_cb_event_t
HF callback events.

*Values:*

enumerator ESP_HF_CONNECTION_STATE_EVT
Connection state changed event

enumerator ESP_HF_AUDIO_STATE_EVT
Audio connection state change event

enumerator ESP_HF_BVRA_RESPONSE_EVT
Voice recognition state change event

enumerator ESP_HF_VOLUME_CONTROL_EVT
Audio volume control command from HF Client, provided by +VGM or +VGS message

enumerator ESP_HF_UNAT_RESPONSE_EVT
Unknown AT cmd Response

enumerator ESP_HF_IND_UPDATE_EVT
Indicator Update Event

enumerator ESP_HF_CIND_RESPONSE_EVT
Call And Device Indicator Response

enumerator ESP_HF_COPS_RESPONSE_EVT
Current operator information

enumerator ESP_HF_CLCC_RESPONSE_EVT
List of current calls notification

enumerator ESP_HF_CNUM_RESPONSE_EVT
Subscriber information response from HF Client

enumerator ESP_HF_VTS_RESPONSE_EVT
Enable or not DTMF

enumerator ESP_HF_NREC_RESPONSE_EVT
Enable or not NREC

enumerator ESP_HF_ATA_RESPONSE_EVT
Answer an Incoming Call
enumerator **ESP_HF_CHUP_RESPONSE_EVT**
  Reject an Incoming Call

enumerator **ESP_HF_DIAL_EVT**
  Origin an outgoing call with specific number or the dial the last number

enumerator **ESP_HF_WBS_RESPONSE_EVT**
  Codec Status

enumerator **ESP_HF_BCS_RESPONSE_EVT**
  Final Codec Choice

**Bluetooth HID Device API**

**Overview**
A Bluetooth HID device is a device providing the service of human or other data input and output to and from a Bluetooth HID Host. Users can use the Bluetooth HID Device APIs to make devices like keyboards, mice, joysticks and so on.

**Application Example**
Check bluetooth/bluedroid/classic_bt folder in ESP-IDF examples, which contains the following application:

- This is an example of Bluetooth HID mouse device. The device running this example can be discovered and connected by a Bluetooth HID Host device such as a PC, and the pointer will move left and right after HID connection is established - bluetooth/bluedroid/classic_bt/bt_hid_mouse_device

**API Reference**

**Header File**
- components/bt/host/bluedroid/api/include/api/esp_hidd_api.h

**Functions**

`esp_err_t esp_bt_hid_device_register_callback (esp_hd_cb_t callback)`
This function is called to init callbacks with HID device module.

参数 `callback` - [in] pointer to the init callback function.

返回
- ESP_OK: success
- other: failed

`esp_err_t esp_bt_hid_device_init (void)`
This function initializes HIDD. This function should be called after esp_bluedroid_enable and esp_bluedroid_init success, and should be called after esp_bt_hid_device_register_callback. When the operation is complete the callback function will be called with ESP_HIDD_INIT_EVT.

返回
- ESP_OK: success
- other: failed

`esp_err_t esp_bt_hid_device_deinit (void)`
This function de-initializes HIDD interface. This function should be called after esp_bluedroid_enable() and esp_bluedroid_init() success, and should be called after esp_bt_hid_device_init(). When the operation is complete the callback function will be called with ESP_HIDD_DEINIT_EVT.

返回 - ESP_OK: success
### Chapter 2. API 参考

- other: failed

```c
esp_err_t esp_bt_hid_device_register_app(esp_hid_app_param_t *app_param,
                                         esp_hid_qos_param_t *in_qos,
                                         esp_hid_qos_param_t *out_qos)
```

Registers HIDD parameters with SDP and sets l2cap Quality of Service. This function should be called after `esp_bluedroid_enable` and `esp_bluedroid_init` success, and must be done after `esp_bt_hid_device_init`. When the operation is complete the callback function will be called with `ESP_HIDD_REGISTER_APP_EVT`.

**参数**
- `app_param` - [in] HIDD parameters
- `in_qos` - [in] incoming QoS parameters
- `out_qos` - [in] outgoing QoS parameters

**返回**
- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_hid_device_unregister_app(void)
```

Removes HIDD parameters from SDP and resets l2cap Quality of Service. This function should be called after `esp_bluedroid_enable` and `esp_bluedroid_init` success, and should be called after `esp_bt_hid_device_init`. When the operation is complete the callback function will be called with `ESP_HIDD_UNREGISTER_APP_EVT`.

**返回**
- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_hid_device_connect(esp_bd_addr_t bd_addr)
```

This function connects HIDD interface to connected bluetooth device, if not done already. When the operation is complete the callback function will be called with `ESP_HIDD_OPEN_EVT`.

**参数**
- `bd_addr` - [in] Remote host bluetooth device address.

**返回**
- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_hid_device_disconnect(void)
```

This function disconnects HIDD interface. When the operation is complete the callback function will be called with `ESP_HIDD_CLOSE_EVT`.

**返回**
- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_hid_device_send_report(esp_hid_report_type_t type, uint8_t tid, uint16_t len,
                                         uint8_t *data)
```

Send HIDD report. When the operation is complete the callback function will be called with `ESP_HIDD_SEND_REPORT_EVT`.

**参数**
- `type` - [in] type of report
- `id` - [in] report id as defined by descriptor
- `len` - [in] length of report
- `data` - [in] report data

**返回**
- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_hid_device_report_error(esp_hid_handshake_error_t error)
```

Sends HIDD handshake with error info for invalid set_report. When the operation is complete the callback function will be called with `ESP_HIDD_REPORT_ERR_EVT`.

**参数**
- `error` - [in] type of error

**返回**
- ESP_OK: success
- other: failed
esp_err_t esp_bt_hid_device_virtual_cable_unplug (void)
    Unplug virtual cable of HIDD. When the operation is complete the callback function will be called with ESP_HIDD_VC_UNPLUG_EVT.
    - ESP_OK: success
    - other: failed

Unions
union esp_hidd_cb_param_t
    #include <esp_hidd_api.h> HID device callback parameters union.

Public Members

struct esp_hidd_cb_param_t::hidd_init_evt_param init
    HIDD callback param of ESP_HIDD_INIT_EVT

struct esp_hidd_cb_param_t::hidd_deinit_evt_param deinit
    HIDD callback param of ESP_HIDD_DEINIT_EVT

struct esp_hidd_cb_param_t::hidd_register_app_evt_param register_app
    HIDD callback param of ESP_HIDD_REGISTER_APP_EVT

struct esp_hidd_cb_param_t::hidd_unregister_app_evt_param unregister_app
    HIDD callback param of ESP_HIDD_UNREGISTER_APP_EVT

struct esp_hidd_cb_param_t::hidd_open_evt_param open
    HIDD callback param of ESP_HIDD_OPEN_EVT

struct esp_hidd_cb_param_t::hidd_close_evt_param close
    HIDD callback param of ESP_HIDD_CLOSE_EVT

struct esp_hidd_cb_param_t::hidd_send_report_evt_param send_report
    HIDD callback param of ESP_HIDD_SEND_REPORT_EVT

struct esp_hidd_cb_param_t::hidd_report_err_evt_param report_err
    HIDD callback param of ESP_HIDD_REPORT_ERR_EVT

struct esp_hidd_cb_param_t::hidd_get_report_evt_param get_report
    HIDD callback param of ESP_HIDD_GET_REPORT_EVT

struct esp_hidd_cb_param_t::hidd_set_report_evt_param set_report
    HIDD callback param of ESP_HIDD_SET_REPORT_EVT

struct esp_hidd_cb_param_t::hidd_set_protocol_evt_param set_protocol
    HIDD callback param of ESP_HIDD_SET_PROTOCOL_EVT

struct esp_hidd_cb_param_t::hidd_intr_data_evt_param intr_data
    HIDD callback param of ESP_HIDD_INTR_DATA_EVT
struct esp_hidd_cb_param_t::hidd_vc_unplug_param vc_unplug
HIDD callback param of ESP_HIDD_VC_UNPLUG_EVT

struct hidd_close_evt_param
#include <esp_hidd_api.h> ESP_HIDD_CLOSE_EVT.

Public Members

esp_hidd_status_t status
operation status

esp_hidd_connection_state_t conn_status
connection status

struct hidd_deinit_evt_param
#include <esp_hidd_api.h> ESP_HIDD_DEINIT_EVT.

Public Members

esp_hidd_status_t status
operation status

struct hidd_get_report_evt_param
#include <esp_hidd_api.h> ESP_HIDD_GET_REPORT_EVT.

Public Members

esp_hidd_report_type_t report_type
report type

uint8_t report_id
report id

uint16_t buffer_size
buffer size

struct hidd_init_evt_param
#include <esp_hidd_api.h> ESP_HIDD_INIT_EVT.

Public Members

esp_hidd_status_t status
operation status

struct hidd_intr_data_evt_param
#include <esp_hidd_api.h> ESP_HIDD_INTR_DATA_EVT.
Public Members

```c
uint8_t report_id
interrupt channel report id
```

```c
uint16_t len
interrupt channel report data length
```

```c
uint8_t* data
interrupt channel report data pointer
```

```c
struct hidd_open_evt_param
#include <esp_hidd_api.h> ESP_HIDD_OPEN_EVT.
```

Public Members

```c
esp_hidd_status_t status
operation status
```

```c
esp_hidd_connection_state_t conn_status
connection status
```

```c
esp_bd_addr_t bd_addr
host address
```

```c
struct hidd_register_app_evt_param
#include <esp_hidd_api.h> ESP_HIDD_REGISTER_APP_EVT.
```

Public Members

```c
esp_hidd_status_t status
operation status
```

```c
bool in_use
indicate whether use virtual cable plug host address
```

```c
esp_bd_addr_t bd_addr
host address
```

```c
struct hidd_report_err_evt_param
#include <esp_hidd_api.h> ESP_HIDD_REPORT_ERR_EVT.
```

Public Members

```c
esp_hidd_status_t status
operation status
```
### Public Members

#### struct hidd_send_report_evt_param

```c
#include <esp_hidd_api.h> ESP_HIDD_SEND_REPORT_EVT.
```

- **esp_hidd_status_t** `status`
  - operation status

- **uint8_t** `reason`
  - lower layer failed reason (ref hiddefs.h)

- **esp_hidd_report_type_t** `report_type`
  - report type

- **uint8_t** `report_id`
  - report id

#### struct hidd_set_protocol_evt_param

```c
#include <esp_hidd_api.h> ESP_HIDD_SET_PROTOCOL_EVT.
```

- **esp_hidd_protocol_mode_t** `protocol_mode`
  - protocol mode

#### struct hidd_set_report_evt_param

```c
#include <esp_hidd_api.h> ESP_HIDD_SET_REPORT_EVT.
```

- **esp_hidd_report_type_t** `report_type`
  - report type

- **uint8_t** `report_id`
  - report id

- **uint16_t** `len`
  - set_report data length

- **uint8_t** `*data`
  - set_report data pointer

#### struct hidd_unregister_app_evt_param

```c
#include <esp_hidd_api.h> ESP_HIDD_UNREGISTER_APP_EVT.
```
Chapter 2. API

Public Members

```c
esp_hidd_status_t status
operation status
```

```c
struct hidd_vc_unplug_param
#include <esp_hidd_api.h> ESP_HIDD_VC_UNPLUG_EVT.
```

Public Members

```c
esp_hidd_status_t status
operation status
```

```c
esp_hidd_connection_state_t conn_status
connection status
```

Structures

struct esp_hidd_app_param_t

HID device characteristics for SDP server.

Public Members

```c
const char *name
service name
```

```c
const char *description
service description
```

```c
const char *provider
provider name
```

```c
uint8_t subclass
HID device subclass
```

```c
uint8_t *desc_list
HID descriptor list
```

```c
int desc_list_len
size in bytes of HID descriptor list
```

struct esp_hidd_qos_param_t

HIDD Quality of Service parameters negotiated over L2CAP.

Public Members
**Chapter 2. API 参考**

```
uint8_t service_type
    the level of service, 0 indicates no traffic

uint32_t token_rate
    token rate in bytes per second, 0 indicates “don’t care”

uint32_t token_bucket_size
    limit on the burstness of the application data

uint32_t peak_bandwidth
    bytes per second, value 0 indicates “don’t care”

uint32_t access_latency
    maximum acceptable delay in microseconds

uint32_t delay_variation
    the difference in microseconds between the max and min delay
```

**Macros**

- **ESP_HID_CLASS_UNKNOWN**
  subclass of hid device
  unknown HID device subclass

- **ESP_HID_CLASS_JOS**
  joystick

- **ESP_HID_CLASS_GPD**
  game pad

- **ESP_HID_CLASS_RMC**
  remote control

- **ESP_HID_CLASS_SED**
  sensing device

- **ESP_HID_CLASS_DGT**
  digitizer tablet

- **ESP_HID_CLASS_CDR**
  card reader

- **ESP_HID_CLASS_KBD**
  keyboard

- **ESP_HID_CLASS_MIC**
  pointing device

- **ESP_HID_CLASS_COM**
  combo keyboard/pointing
Type Definitions

typedef void (*esp_hd_cb_t)(esp_hidd_cb_event_t event, esp_hidd_cb_param_t *param)

HID device callback function type.

- **Param event**: Event type
- **Param param**: Point to callback parameter, currently is union type

Enumerations

enum esp_hidd_handshake_error_t

HIDD handshake result code.

- **enumerator ESP_HID_PAR_HANDSHAKE_RSP_SUCCESS**: successful
- **enumerator ESP_HID_PAR_HANDSHAKE_RSP_NOT_READY**: not ready, device is too busy to accept data
- **enumerator ESP_HID_PAR_HANDSHAKE_RSP_ERR_INVALID_REP_ID**: invalid report ID
- **enumerator ESP_HID_PAR_HANDSHAKE_RSP_ERR_UNSUPPORTED_REQ**: device does not support the request
- **enumerator ESP_HID_PAR_HANDSHAKE_RSP_ERR_INVALID_PARAM**: parameter value is out of range or inappropriate
- **enumerator ESP_HID_PAR_HANDSHAKE_RSP_ERR_UNKNOWN**: device could not identify the error condition
- **enumerator ESP_HID_PAR_HANDSHAKE_RSP_ERR_FATAL**: restart is essential to resume functionality

enum esp_hidd_report_type_t

HIDD report types.

- **enumerator ESP_HID_REPORT_TYPE_OTHER**: unknown report type
- **enumerator ESP_HID_REPORT_TYPE_INPUT**: input report
- **enumerator ESP_HID_REPORT_TYPE_OUTPUT**: output report
- **enumerator ESP_HID_REPORT_TYPE_FEATURE**: feature report
enumerator ESP_HIDD_REPORT_TYPE_INTRDATA
    special value for reports to be sent on interrupt channel, INPUT is assumed

enum esp_hidd_connection_state_t
    HIDD connection state.
    Values:

    enumerator ESP_HIDD_CONN_STATE_CONNECTED
        HID connection established

    enumerator ESP_HIDD_CONN_STATE_CONNECTING
        connection to remote Bluetooth device

    enumerator ESP_HIDD_CONN_STATE_DISCONNECTED
        connection released

    enumerator ESP_HIDD_CONN_STATE_DISCONNECTING
        disconnecting to remote Bluetooth device

    enumerator ESP_HIDD_CONN_STATE_UNKNOWN
        unknown connection state

enum esp_hidd_protocol_mode_t
    HID device protocol modes.
    Values:

    enumerator ESP_HIDD_REPORT_MODE
        Report Protocol Mode

    enumerator ESP_HIDD_BOOT_MODE
        Boot Protocol Mode

    enumerator ESP_HIDD_UNSUPPORTED_MODE
        unsupported

enum esp_hidd_boot_report_id_t
    HID Boot Protocol report IDs.
    Values:

    enumerator ESP_HIDD_BOOT_REPORT_ID_KEYBOARD
        report ID of Boot Protocol keyboard report

    enumerator ESP_HIDD_BOOT_REPORT_ID_MOUSE
        report ID of Boot Protocol mouse report

enum [anonymous]
    HID Boot Protocol report size including report ID.
    Values:
Chapter 2. API 參考

enumerator **ESP_HIDD_BOOT_REPORT_SIZE_KEYBOARD**
report size of Boot Protocol keyboard report

enumerator **ESP_HIDD_BOOT_REPORT_SIZE_MOUSE**
report size of Boot Protocol mouse report

eNUM **esp_hidd_cb_event_t**
HID device callback function events.
Values:

enumerator **ESP_HIDD_INIT_EVT**
When HID device is initialized, the event comes

enumerator **ESP_HIDD_DEINIT_EVT**
When HID device is deinitialized, the event comes

enumerator **ESP_HIDD_REGISTER_APP_EVT**
When HID device application registered, the event comes

enumerator **ESP_HIDD_UNREGISTER_APP_EVT**
When HID device application unregistered, the event comes

enumerator **ESP_HIDD_OPEN_EVT**
When HID device connection to host opened, the event comes

enumerator **ESP_HIDD_CLOSE_EVT**
When HID device connection to host closed, the event comes

enumerator **ESP_HIDD_SEND_REPORT_EVT**
When HID device send report to lower layer, the event comes

enumerator **ESP_HIDD_REPORT_ERR_EVT**
When HID device report handshake error to lower layer, the event comes

enumerator **ESP_HIDD_GET_REPORT_EVT**
When HID device receives GET_REPORT request from host, the event comes

enumerator **ESP_HIDD_SET_REPORT_EVT**
When HID device receives SET_REPORT request from host, the event comes

enumerator **ESP_HIDD_SET_PROTOCOL_EVT**
When HID device receives SET_PROTOCOL request from host, the event comes

enumerator **ESP_HIDD_INTR_DATA_EVT**
When HID device receives DATA from host on intr, the event comes

enumerator **ESP_HIDD_VC_UNPLUG_EVT**
When HID device initiates Virtual Cable Unplug, the event comes
enumerator **ESP_HIDD_API_ERR_EVT**

When HID device has API error, the event comes

enum **esp_hidd_status_t**

Values:

enumerator **ESP_HIDD_SUCCESS**

enumerator **ESP_HIDD_ERROR**

general ESP HD error

enumerator **ESP_HIDD_NO_RES**

out of system resources

enumerator **ESP_HIDD_BUSY**

Temporarily can not handle this request.

enumerator **ESP_HIDD_NO_DATA**

No data.

enumerator **ESP_HIDD_NEED_INIT**

HIDD module shall init first

enumerator **ESP_HIDD_NEED_DEINIT**

HIDD module shall deinit first

enumerator **ESP_HIDD_NEED_REG**

HIDD module shall register first

enumerator **ESP_HIDD_NEED_DEREG**

HIDD module shall deregister first

enumerator **ESP_HIDD_NO_CONNECTION**

connection may have been closed

### Classic Bluetooth L2CAP API

**Application Example** Check bluetooth/bluedroid/classic_bt folder in ESP-IDF examples, which contains the following application:

• This is a BT_L2CAP demo. This demo can connect, send and receive L2CAP data bluetooth/bluedroid/classic_bt/bt_l2cap_client, bluetooth/bluedroid/classic_bt/bt_l2cap_server

**API Reference**

**Header File**

• components/bt/host/bluedroid/api/include/api/esp_l2cap_bt_api.h
Chapter 2. API 参考

Functions

**esp_err_t esp bt l2cap_register_callback (esp bt l2cap cb f callback)**

This function is called to init callbacks with L2CAP module.

- 参数 callback - [in] pointer to the init callback function.
- 返回 • ESP_OK: success
  • other: failed

**esp_err_t esp bt l2cap_init (void)**

This function is called to init L2CAP module. When the operation is completed, the callback function will be called with ESP_BT_L2CAP_INIT_EVT. This function should be called after esp_bluedroid_enable() completes successfully.

- 返回 • ESP_OK: success
  • other: failed

**esp_err_t esp bt l2cap_deinit (void)**

This function is called to uninit l2cap module. The operation will close all active L2CAP connection first, then the callback function will be called with ESP_BT_L2CAP_CLOSE_EVT, and the number of ESP_BT_L2CAP_CLOSE_EVT is equal to the number of connection. When the operation is completed, the callback function will be called with ESP_BT_L2CAP_UNINIT_EVT. This function should be called after esp bt l2cap_init() completes successfully.

- 返回 • ESP_OK: success
  • other: failed

**esp_err_t esp bt l2cap_connect (esp bt l2cap cntl flags t cntl_flag, uint16_t remote_psm, esp bd addr t peer_bd_addr)**

This function makes an L2CAP connection to a remote BD Address. When the connection is initiated or failed to initiate, the callback is called with ESP_BT_L2CAP_CL_INIT_EVT. When the connection is established or failed, the callback is called with ESP_BT_L2CAP_OPEN_EVT. This function must be called after esp bt l2cap_init() successful and before esp bt l2cap_deinit().

- 参数 • cntl_flag - [in] Lower 16-bit security settings mask.
  • remote_psm - [in] Remote device bluetooth Profile PSM.
  • peer_bd_addr - [in] Remote device bluetooth device address.
- 返回 • ESP_OK: success
  • other: failed

**esp_err_t esp bt l2cap_start_srv (esp bt l2cap cntl flags t cntl_flag, uint16_t local_psm)**

This function create a L2CAP server and starts listening for an L2CAP connection request from a remote Bluetooth device. When the server is started successfully, the callback is called with ESP_BT_L2CAP_START_EVT. When the connection is established, the callback is called with ESP_BT_L2CAP_OPEN_EVT. This function must be called after esp bt l2cap_init() successful and before esp bt l2cap_deinit().

- 参数 • cntl_flag - [in] Lower 16-bit security settings mask.
  • local_psm - [in] Dynamic PSM.
- 返回 • ESP_OK: success
  • other: failed

**esp_err_t esp bt l2cap_stop_all_srv (void)**

This function stops all L2CAP servers. The operation will close all active L2CAP connection first, then the callback function will be called with ESP_BT_L2CAP_CLOSE_EVT, and the number of
ESP_BT_L2CAP_CLOSE_EVT is equal to the number of connection. When the operation is completed, the callback is called with ESP_BT_L2CAP_SRV_STOP_EVT. This function must be called after esp_bt_l2cap_init() successful and before esp_bt_l2cap_deinit().

- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_l2cap_stop_srv(uint16_t local_psm)
```
This function stops a specific L2CAP server. The operation will close all active L2CAP connection first on the specific L2CAP server, then the callback function will be called with ESP_BT_L2CAP_CLOSE_EVT, and the number of ESP_BT_L2CAP_CLOSE_EVT is equal to the number of connection. When the operation is completed, the callback is called with ESP_BT_L2CAP_SRV_STOP_EVT. This function must be called after esp_bt_l2cap_init() successful and before esp_bt_l2cap_deinit().

- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_l2cap_vfs_register(void)
```
This function is used to register VFS. Only supports write, read and close. This function must be called after esp_bt_l2cap_init() successful and before esp_bt_l2cap_deinit().

- ESP_OK: success
- other: failed

```c
esp_err_t esp_bt_l2cap_vfs_unregister(void)
```
This function is used to unregister VFS. This function must be called after esp_bt_l2cap_init() successful and before esp_bt_l2cap_deinit().

- ESP_OK: success
- other: failed

**Unions**

```c
union esp_bt_l2cap_cb_param_t
```

- `#include <esp_l2cap_bt_api.h>` L2CAP callback parameters union.

**Public Members**

```c
struct esp_bt_l2cap_cb_param_t::l2cap_init_evt_param init
L2CAP callback param of ESP_BT_L2CAP_INIT_EVT
```

```c
struct esp_bt_l2cap_cb_param_t::l2cap_uninit_evt_param uninit
L2CAP callback param of ESP_BT_L2CAP_UNINIT_EVT
```

```c
struct esp_bt_l2cap_cb_param_t::l2cap_open_evt_param open
L2CAP callback param of ESP_BT_L2CAP_OPEN_EVT
```

```c
struct esp_bt_l2cap_cb_param_t::l2cap_close_evt_param close
L2CAP callback param of ESP_BT_L2CAP_CLOSE_EVT
```
struct esp_bt_l2cap_cb_param_t::l2cap_start_evt_param\textbf{ start}\n
L2CAP callback param of ESP_BT_L2CAP_START_EVT

struct esp_bt_l2cap_cb_param_t::l2cap_cl_init_evt_param\textbf{ cl_init}\n
L2CAP callback param of ESP_BT_L2CAP_CL_INIT_EVT

struct esp_bt_l2cap_cb_param_t::l2cap_srv_stop_evt_param\textbf{ srv_stop}\n
L2CAP callback param of ESP_BT_L2CAP_SRV_STOP_EVT

struct l2cap_cl_init_evt_param\n
\#include \textless esp\_l2cap\_bt\_api.h\textgreater ESP_BT_L2CAP_CL_INIT_EVT.

\textbf{Public Members}\n
\textit{esp\_bt\_l2cap\_status\_t status}\n
status

\textit{uint32\_t handle}\n
The connection handle

\textit{uint8\_t sec\_id}\n
security ID used by this server

struct l2cap_close_evt_param\n
\#include \textless esp\_l2cap\_bt\_api.h\textgreater ESP_BT_L2CAP_CLOSE_EVT.

\textbf{Public Members}\n
\textit{esp\_bt\_l2cap\_status\_t status}\n
status

\textit{uint32\_t handle}\n
The connection handle

\textit{bool async}\n
FALSE, if local initiates disconnect

struct l2cap_init_evt_param\n
\#include \textless esp\_l2cap\_bt\_api.h\textgreater ESP_BT_L2CAP_INIT_EVT.

\textbf{Public Members}\n
\textit{esp\_bt\_l2cap\_status\_t status}\n
status

struct l2cap_open_evt_param\n
\#include \textless esp\_l2cap\_bt\_api.h\textgreater ESP_BT_L2CAP_OPEN_EVT.
Chapter 2. API

### Public Members

```c
esp_bt_l2cap_status_t status
```

- `status`

```c
uint32_t handle
```

- The connection handle

```c
int fd
```

- File descriptor

```c
esp_bd_addr_t rem_bda
```

- The peer address

```c
int32_t tx_mtu
```

- The transmit MTU

#### struct `l2cap_srv_stop_evt_param`

- Include `<esp_l2cap_bt_api.h>` ESP_BT_L2CAP_SRV_STOP_EVT.

### Public Members

```c
esp_bt_l2cap_status_t status
```

- `status`

```c
uint8_t psm
```

- `psm`

#### struct `l2cap_start_evt_param`

- Include `<esp_l2cap_bt_api.h>` ESP_BT_L2CAP_START_EVT.

### Public Members

```c
esp_bt_l2cap_status_t status
```

- `status`

```c
uint32_t handle
```

- The connection handle

```c
uint8_t sec_id
```

- Security ID used by this server

#### struct `l2cap_uninit_evt_param`

- Include `<esp_l2cap_bt_api.h>` ESP_BT_L2CAP_UNINIT_EVT.
Public Members

```c
esp_bt_l2cap_status_t status
```

Macros

```c
ESP_BT_L2CAP_SEC_NONE
```
Security Setting Mask. Use these three mask mode:

- a. ESP_BT_L2CAP_SEC_NONE
- b. ESP_BT_L2CAP_SEC_AUTHENTICATE
- c. (ESP_BT_L2CAP_SEC_ENCRYPT|ESP_BT_L2CAP_SEC_AUTHENTICATE) No security

```c
ESP_BT_L2CAP_SEC_AUTHORIZE
```
Authorization required

```c
ESP_BT_L2CAP_SEC_AUTHENTICATE
```
Authentication required

```c
ESP_BT_L2CAP_SEC_ENCRYPT
```
Encryption required

Type Definitions

```c
typedef uint32_t esp_bt_l2cap_ctrl_flags_t

typedef void(* esp_bt_l2cap_cb_t)(esp_bt_l2cap_ctrl_event_t event, esp_bt_l2cap_cb_param_t *param)
```
L2CAP callback function type.

- **Param event** Event type
- **Param param** Point to callback parameter, currently is union type

Enumerations

```c
enum esp_bt_l2cap_status_t
```
L2CAP operation success and failure codes.

**Values**:

- enumerator ESP_BT_L2CAP_SUCCESS
  Successful operation.

- enumerator ESP_BT_L2CAP_FAILURE
  Generic failure.

- enumerator ESP_BT_L2CAP_BUSY
  Temporarily can not handle this request.

- enumerator ESP_BT_L2CAP_NO_RESOURCE
  No more resource
enumerator **ESP_BT_L2CAP_NEED_INIT**
L2CAP module shall init first

enumerator **ESP_BT_L2CAP_NEED_DEINIT**
L2CAP module shall deinit first

enumerator **ESP_BT_L2CAP_NO_CONNECTION**
Connection may have been closed

enumerator **ESP_BT_L2CAP_NO_SERVER**
No server

enum **esp_bt_l2cap_cb_event_t**
L2CAP callback function events.

*Values:*

enumerator **ESP_BT_L2CAP_INIT_EVT**
When L2CAP is initialized, the event comes

enumerator **ESP_BT_L2CAP_UNINIT_EVT**
When L2CAP is deinitialized, the event comes

enumerator **ESP_BT_L2CAP_OPEN_EVT**
When L2CAP Client connection open, the event comes

enumerator **ESP_BT_L2CAP_CLOSE_EVT**
When L2CAP connection closed, the event comes

enumerator **ESP_BT_L2CAP_START_EVT**
When L2CAP server started, the event comes

enumerator **ESP_BT_L2CAP_CL_INIT_EVT**
When L2CAP client initiated a connection, the event comes

enumerator **ESP_BT_L2CAP_SRV_STOP_EVT**
When L2CAP server stopped, the event comes

**BT SDP APIs**

**Overview**  Bluetooth SDP reference APIs.

**API Reference**

**Header File**

- components/bt/host/bluedroid/api/include/api/esp_sdp_api.h
Functions

`esp_err_t esp_sdp_register_callback (esp_sdp_cb_t callback)`
This function is called to init callbacks with SDP module.

参数 `callback` [in] pointer to the init callback function.

返回
  • ESP_OK: success
  • other: failed

`esp_err_t esp_sdp_init (void)`
This function is called to init SDP module. When the operation is completed, the callback function will be called with ESP_SDP_INIT_EVT. This function should be called after `esp_bluedroid_enable()` completes successfully.

返回
  • ESP_OK: success
  • other: failed

`esp_err_t esp_sdp_deinit (void)`
This function is called to de-initialize SDP module. The operation will remove all SDP records, then the callback function will be called with ESP_SDP_REMOVE_RECORD_COMP_EVT, and the number of ESP_SDP_REMOVE_RECORD_COMP_EVT is equal to the number of SDP records. When the operation is completed, the callback function will be called with ESP_SDP_DEINIT_EVT. This function should be called after `esp_sdp_init()` completes successfully.

返回
  • ESP_OK: success
  • other: failed

`esp_err_t esp_sdp_search_record (esp_bd_addr_t bd_addr, esp_bt_uuid_t uuid)`
This function is called to perform service discovery for the services provided by the given peer device. When the operation is completed, the callback function will be called with ESP_SDP_SEARCH_COMP_EVT. This function must be called after `esp_sdp_init()` successful and before `esp_sdp_deinit()`.

参数
  • `bd_addr` [in] Remote device bluetooth device address.
  • `uuid` [in] Service UUID of the remote device.

返回
  • ESP_OK: success
  • other: failed

`esp_err_t esp_sdp_create_record (esp_bluetooth_sdp_record_t *record)`
This function is called to create SDP records. When the operation is completed, the callback function will be called with ESP_SDP_CREATE_RECORD_COMP_EVT. This function must be called after `esp_sdp_init()` successful and before `esp_sdp_deinit()`.

参数 `record` [in] The SDP record to create.

返回
  • ESP_OK: success
  • other: failed

`esp_err_t esp_sdp_remove_record (int record_handle)`
This function is called to remove a SDP record. When the operation is completed, the callback function will be called with ESP_SDP_REMOVE_RECORD_COMP_EVT. This function must be called after `esp_sdp_init()` successful and before `esp_sdp_deinit()`.

参数 `record_handle` [in] The SDP record handle.

返回
  • ESP_OK: success
  • other: failed

Unions
union esp_bluetooth_sdp_record_t

#include <esp_sdp_api.h> SDP record parameters union.

Public Members

ext_bluetooth_sdp_hdr_overlay_t hdr
  General info

est_bluetooth_sdp_mas_record_t mas
  Message Access Profile - Server

est_bluetooth_sdp_mns_record_t mns
  Message Access Profile - Client (Notification Server)

est_bluetooth_sdp_pse_record_t pse
  Phone Book Profile - Server

est_bluetooth_sdp_pce_record_t pce
  Phone Book Profile - Client

est_bluetooth_sdp_ops_record_t ops
  Object Push Profile

est_bluetooth_sdp_sap_record_t sap
  SIM Access Profile

union esp_sdp_cb_param_t

#include <esp_sdp_api.h> SDP callback parameters union.

Public Members

struct esp_sdp_cb_param_t::sdp_init_evt_param init
  SDP callback param of ESP_SDP_INIT_EVT

struct esp_sdp_cb_param_t::sdp_deinit_evt_param deinit
  SDP callback param of ESP_SDP_DEINIT_EVT

struct esp_sdp_cb_param_t::sdp_search_evt_param search
  SDP callback param of ESP_SDP_SEARCH_COMP_EVT

struct esp_sdp_cb_param_t::sdp_crate_record_evt_param create_record
  SDP callback param of ESP_SDP_CREATE_RECORD_COMP_EVT

struct esp_sdp_cb_param_t::sdp_remove_record_evt_param remove_record
  SDP callback param of ESP_SDP_REMOVE_RECORD_COMP_EVT

struct sdp_crate_record_evt_param

#include <esp_sdp_api.h> ESP_SDP_CREATE_RECORD_COMP_EVT.
Public Members

```c
esp_sdp_status_t status
```
status

```c
int record_handle
```
SDP record handle

```c
struct sdp_deinit_evt_param
```

```
#include <esp_sdp_api.h>
ESP_SDP_DEINIT_EVT.
```

Public Members

```c
esp_sdp_status_t status
```
status

```c
struct sdp_init_evt_param
```

```
#include <esp_sdp_api.h>
ESP_SDP_INIT_EVT.
```

Public Members

```c
esp_sdp_status_t status
```
status

```c
struct sdp_remove_record_evt_param
```

```
#include <esp_sdp_api.h>
ESP_SDP_REMOVE_RECORD_COMP_EVT.
```

Public Members

```c
esp_sdp_status_t status
```
status

```c
struct sdp_search_evt_param
```

```
#include <esp_sdp_api.h>
ESP_SDP_SEARCH_COMP_EVT.
```

Public Members

```c
esp_sdp_status_t status
```
status

```c
esp_bd_addr_t remote_addr
```
remote device address

```c
esp_bt_uuid_t sdp_uuid
```
service uuid
Chapter 2. API 参考

```c
int record_count
    Number of SDP records

esp.bluetooth_sdp_record_t *records
    SDP records
```

**Structures**

struct `bluetooth_sdp_hdr_overlay`

Some signals need additional pointers, hence we introduce a generic way to handle these pointers.

**Public Members**

```c
esp.bluetooth_sdp_types_t type
    SDP type

esp.bt_uuid_t uuid
    UUID type, include uuid and uuid length

uint32_t service_name_length
    Service name length

t char *service_name
    service name

int32_t rfcomm_channel_number
    rfcomm channel number, if not used set to -1

int32_t l2cap_psm
    l2cap psm, if not used set to -1

int32_t profile_version
    profile version

int user1_ptr_len
    see esp.bluetooth_sdp_ops_record_t

uint8_t *user1_ptr
    see esp.bluetooth_sdp_ops_record_t

int user2_ptr_len
    see esp.bluetooth_sdp_ops_record_t

uint8_t *user2_ptr
    see esp.bluetooth_sdp_ops_record_t
```

struct `bluetooth_sdp_mas_record`

Message Access Profile - Server parameters.
Public Members

`exp_bluetooth_sdp_hdr_overlay_t hdr`
General info

`uint32_t mas_instance_id`
MAS Instance ID

`uint32_t supported_features`
Map supported features

`uint32_t supported_message_types`
Supported message types

`struct bluetooth_sdp_mns_record`
Message Access Profile - Client (Notification Server) parameters.

Public Members

`exp_bluetooth_sdp_hdr_overlay_t hdr`
General info

`uint32_t supported_features`
Supported features

`struct bluetooth_sdp_pse_record`
Phone Book Profile - Server parameters.

Public Members

`exp_bluetooth_sdp_hdr_overlay_t hdr`
General info

`uint32_t supported_features`
Phap Supported Features

`uint32_t supported_repositories`
Supported Repositories

`struct bluetooth_sdp_pce_record`
Phone Book Profile - Client parameters.

Public Members

`exp_bluetooth_sdp_hdr_overlay_t hdr`
General info
struct bluetooth_sdp_ops_record
Object Push Profile parameters.

Public Members

*esp_bluetooth_sdp_hdr_overlay_t* hdr
General info

int supported_formats_list_len
Supported formats list length

uint8_t *supported_formats_list[SDP_OPP_SUPPORTED_FORMATS_MAX_LENGTH]
Supported formats list

struct bluetooth_sdp_sap_record
SIM Access Profile parameters.

Public Members

*esp_bluetooth_sdp_hdr_overlay_t* hdr
General info

 Macros

ESP_SDP_SERVER_NAME_MAX
Service name max length

SDP_OPP_SUPPORTED_FORMATS_MAX_LENGTH
OPP supported format list maximum length

Type Definitions
typedef struct bluetooth_sdp_hdr_overlay esp_bluetooth_sdp_hdr_overlay_t
Some signals need additional pointers, hence we introduce a generic way to handle these pointers.
typedef struct bluetooth_sdp_mas_record esp_bluetooth_sdp_mas_record_t
Message Access Profile - Server parameters.
typedef struct bluetooth_sdp_mns_record esp_bluetooth_sdp_mns_record_t
Message Access Profile - Client (Notification Server) parameters.
typedef struct bluetooth_sdp_pse_record esp_bluetooth_sdp_pse_record_t
Phone Book Profile - Server parameters.
typedef struct bluetooth_sdp_pce_record esp_bluetooth_sdp_pce_record_t
Phone Book Profile - Client parameters.
typedef struct bluetooth_sdp_ops_record esp_bluetooth_sdp_ops_record_t
  Object Push Profile parameters.

typedef struct bluetooth_sdp_sap_record esp_bluetooth_sdp_sap_record_t
  SIM Access Profile parameters.

typedef void (*esp_sdp_cb_t)(esp_sdp_cb_event_t event, esp_sdp_cb_param_t *param)
  SDP callback function type.
    Param event  Event type
    Param param  Point to callback parameter, currently is union type

Enumerations

enum esp_sdp_status_t
  Values:
    enumerator ESP_SDP_SUCCESS
      Successful operation.
    enumerator ESP_SDP_FAILURE
      Generic failure.
    enumerator ESP_SDP_NO_RESOURCE
      No more resource
    enumerator ESP_SDP_NEED_INIT
      SDP module shall init first
    enumerator ESP_SDP_NEED_DEINIT
      SDP module shall deinit first
    enumerator ESP_SDP_NO_CREATE_RECORD
      No record created

enum esp_sdp_cb_event_t
  SDP callback function events.
  Values:
    enumerator ESP_SDP_INIT_EVT
      When SDP is initialized, the event comes
    enumerator ESP_SDP_DEINIT_EVT
      When SDP is deinitialized, the event comes
    enumerator ESP_SDP_SEARCH_COMP_EVT
      When SDP search complete, the event comes
    enumerator ESP_SDP_CREATE_RECORD_COMP_EVT
      When create SDP records complete, the event comes
### 2.3.4 Controller && VHCI

#### Application Example

Check `bluetooth/hci` folder in ESP-IDF examples, which contains the following application:

- This is a BLE advertising demo with virtual HCI interface. Send `reset/ADV_PARAM/ADV_DATA/ADV_ENABLE` HCI command for BLE advertising - `bluetooth/hci/controller_vhci_ble_adv`.

#### API Reference

**Header File**

- `components/bt/include/esp32/include/esp_bt.h`

**Functions**

```c
esp_err_t esp_ble_tx_power_set (esp_ble_power_type_t power_type, esp_power_level_t power_level)
```

Set BLE TX power Connection Tx power should only be set after connection created.

**参数**

- `power_type`: The type of which tx power, could set Advertising/Connection/Default and etc
• `power_level` - Power level(index) corresponding to absolute value(dbm)

```
esp_power_level_t esp_ble_tx_power_get (esp_power_type_t power_type)
```

Get BLE TX power Connection Tx power should only be get after connection created.

参数 `power_type` - The type of which tx power, could set Advertising/Connection/Default and etc

返回 `>= 0` - Power level, `< 0` - Invalid

```
esp_err_t esp_bredr_tx_power_set (esp_power_type_t power_type, esp_power_level_t min_power_level, esp_power_level_t max_power_level)
```

Set BR/EDR TX power BR/EDR power control will use the power in range of minimum value and maximum value. The power level will effect the global BR/EDR TX power, such inquire, page, connection and so on. Please call the function after `esp_bt_controller_enable` and before any function which cause RF do TX. So you can call the function before doing discovery, profile init and so on. For example, if you want BR/EDR use the new TX power to do inquire, you should call this function before inquire. Another word, If call this function when BR/EDR is in inquire(ING), please do inquire again after call this function. Default minimum power level is ESP_PWR_LVL_N0, and maximum power level is ESP_PWR_LVL_P3.

参数

• `min_power_level` - The minimum power level
• `max_power_level` - The maximum power level

返回 `ESP_OK` - success, other - failed

```
esp_err_t esp_bredr_tx_power_get (esp_power_type_t power_type, esp_power_level_t *min_power_level, esp_power_level_t *max_power_level)
```

Get BR/EDR TX power If the argument is not NULL, then store the corresponding value.

参数

• `min_power_level` - The minimum power level
• `max_power_level` - The maximum power level

返回 `ESP_OK` - success, other - failed

```
esp_err_t esp_bredr_sco_datapath_set (esp_sco_data_path_t data_path)
```

Set default SCO datapath Should be called after controller is enabled, and before (e)SCO link is established.

参数 `data_path` - SCO data path

返回 `ESP_OK` - success, other - failed

```
esp_err_t esp_bt_controller_init (esp_bt_controller_config_t *cfg)
```

Initialize BT controller to allocate task and other resource. This function should be called only once, before any other BT functions are called.

参数 `cfg` - Initial configuration of BT controller. Different from previous version, there’s a mode and some connection configuration in “cfg” to configure controller work mode and allocate the resource which is needed.

返回 `ESP_OK` - success, other - failed

```
esp_err_t esp_bt_controller_deinit (void)
```

De-initialize BT controller to free resource and delete task. You should stop advertising and scanning, as well as disconnect all existing connections before de-initializing BT controller.

This function should be called only once, after any other BT functions are called.

返回 `ESP_OK` - success, other - failed

```
esp_err_t esp_bt_controller_enable (esp_bt_mode_t mode)
```

Enable BT controller. Due to a known issue, you cannot call `esp_bt_controller_enable()` a second time to change the controller mode dynamically. To change controller mode, call `esp_bt_controller_disable()` and then call `esp_bt_controller_enable()` with the new mode.

参数 `mode` - the mode(BLE/BT/BTDI) to enable. For compatible of API, retain this argument.

This mode must be equal as the mode in “cfg” of `esp_bt_controller_init()`. 
## Chapter 2. API 参考

### esp_err_t esp_bt_controller_disable (void)
Disable BT controller.

### esp_bt_controller_status_t esp_bt_controller_get_status (void)
Get BT controller is initialised/de-initialised/enabled/disabled.

### bool esp_vhci_host_check_send_available (void)
esp_vhci_host_check_send_available used for check actively if the host can send packet to controller or not.

### void esp_vhci_host_send_packet (uint8_t *data, uint16_t len)
esp_vhci_host_send_packet host send packet to controller

Should not call this function from within a critical section or when the scheduler is suspended.

#### 参数
- **data** - the packet point
- **len** – the packet length

### esp_err_t esp_vhci_host_register_callback (const esp_vhci_host_callback_t *callback)
esp_vhci_host_register_callback register the vhci reference callback struct defined by vhci_host_callback structure.

#### 参数
- **callback** - esp_vhci_host_callback type variable

### esp_err_t esp_bt_controller_mem_release (esp_bt_mode_t mode)
esp_bt_controller_mem_release release the controller memory as per the mode

This function releases the BSS, data and other sections of the controller to heap. The total size is about 70k bytes.

esp_bt_controller_mem_release(mode) should be called only before esp_bt_controller_init() or after esp_bt_controller_deinit().

Note that once BT controller memory is released, the process cannot be reversed. It means you cannot use the bluetooth mode which you have released by this function.

If your firmware will later upgrade the Bluetooth controller mode (BLE -> BT Classic or disabled -> enabled) then do not call this function.

If the app calls esp_bt_controller_enable(ESP_BT_MODE_BLE) to use BLE only then it is safe to call esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT) at initialization time to free unused BT Classic memory.

If the mode is ESP_BT_MODE_BTDM, then it may be useful to call API esp_bt_mem_release(ESP_BT_MODE_BTDM) instead, which internally calls esp_bt_controller_mem_release(ESP_BT_MODE_BTDM) and additionally releases the BSS and data consumed by the BT/BLE host stack to heap. For more details about usage please refer to the documentation of esp_bt_mem_release() function.

#### 参数
- **mode** - the mode want to release memory

### esp_err_t esp_bt_mem_release (esp_bt_mode_t mode)
esp_bt_mem_release release controller memory and BSS and data section of the BT/BLE host stack as per the mode
This function first releases controller memory by internally calling esp_bt_controller_mem_release(). Additionally, if the mode is set to ESP_BT_MODE_BTDM, it also releases the BSS and data consumed by the BT/BLE host stack to heap.

Note that once BT memory is released, the process cannot be reversed. It means you cannot use the bluetooth mode which you have released by this function.

If your firmware will later upgrade the Bluetooth controller mode (BLE -> BT Classic or disabled -> enabled) then do not call this function.

If you never intend to use bluetooth in a current boot-up cycle, you can call esp_bt_mem_release(ESP_BT_MODE_BTDM) before esp_bt_controller_init or after esp_bt_controller_deinit.

For example, if a user only uses bluetooth for setting the WiFi configuration, and does not use bluetooth in the rest of the product operation. In such cases, after receiving the WiFi configuration, you can disable/deinit bluetooth and release its memory. Below is the sequence of APIs to be called for such scenarios:

```c
esp_bluedroid_disable();
esp_bluedroid_deinit();
esp_bt_controller_disable();
esp_bt_controller_deinit();
esp_bt_mem_release(ESP_BT_MODE_BTDM);
```

备注： In case of NimBLE host, to release BSS and data memory to heap, the mode needs to be set to ESP_BT_MODE_BTDM as controller is dual mode.

### esp_err_t esp_bt_sleep_enable(void)

enable bluetooth to enter modem sleep

Note that this function shall not be invoked before esp_bt_controller_enable().

There are currently two options for bluetooth modem sleep, one is ORIG mode, and another is EVED Mode. EVED Mode is intended for BLE only.

For ORIG mode: Bluetooth modem sleep is enabled in controller start up by default if CONFIG_CTRL_BTDM_MODEM_SLEEP is set and "ORIG mode" is selected. In ORIG modem sleep mode, bluetooth controller will switch off some components and pause to work every now and then, if there is no event to process; and wakeup according to the scheduled interval and resume the work. It can also wakeup earlier upon external request using function "esp_bt_controller_wakeup_request".

返回
- ESP_OK: success
- other: failed

### esp_err_t esp_bt_sleep_disable (void)

disable bluetooth modem sleep

Note that this function shall not be invoked before esp_bt_controller_enable().

If esp_bt_sleep_disable() is called, bluetooth controller will not be allowed to enter modem sleep.

If ORIG modem sleep mode is in use, if this function is called, bluetooth controller may not immediately wake up if it is dormant then. In this case, esp_bt_controller_wakeup_request() can be used to shorten the time for wakeup.

返回
- ESP_OK: success
- other: failed
**esp_err_t esp_ble_scan_duplicated_list_flush (void)**  
Manually clear scan duplicate list.  
Note that scan duplicate list will be automatically cleared when the maximum amount of device in the filter is reached the amount of device in the filter can be configured in menuconfig.

*备注：* This function name is incorrectly spelled, it will be fixed in release 5.x version.

- **返回**  
  - ESP_OK: success  
  - other: failed

**void esp_wifi_bt_power_domain_on (void)**  
bt Wi-Fi power domain power on

**void esp_wifi_bt_power_domain_off (void)**  
bt Wi-Fi power domain power off

**Structures**

- **struct esp_bt_controller_config_t**  
Controller config options, depend on config mask. Config mask indicate which functions enabled, this means some options or parameters of some functions enabled by config mask.

**Public Members**

- **uint16_t controller_task_stack_size**  
  Bluetooth controller task stack size

- **uint8_t controller_task_prio**  
  Bluetooth controller task priority

- **uint8_t hci_uart_no**  
  If use UART1/2 as HCI IO interface, indicate UART number

- **uint32_t hci_uart_baudrate**  
  If use UART1/2 as HCI IO interface, indicate UART baudrate

- **uint8_t scan_duplicate_mode**  
  scan duplicate mode

- **uint8_t scan_duplicate_type**  
  scan duplicate type

- **uint16_t normal_adv_size**  
  Normal adv size for scan duplicate

- **uint16_t mesh_adv_size**  
  Mesh adv size for scan duplicate
uint16_t send_adv_reserved_size
Controller minimum memory value

uint32_t controller_debug_flag
Controller debug log flag

uint8_t mode
Controller mode: BR/EDR, BLE or Dual Mode

uint8_t ble_max_conn
BLE maximum connection numbers

uint8_t bt_max_acl_conn
BR/EDR maximum ACL connection numbers

uint8_t bt_sco_datapath
SCO data path, i.e. HCI or PCM module

bool auto_latency
BLE auto latency, used to enhance classic BT performance

bool bt_legacy_auth_vs_evt
BR/EDR Legacy auth complete event required to protect from BIAS attack

uint8_t bt_max_sync_conn
BR/EDR maximum ACL connection numbers. Effective in menuconfig

uint8_t ble_sca
BLE low power crystal accuracy index

uint8_t pcm_role
PCM role (master & slave)

uint8_t pcm_polar
PCM polar trig (falling clk edge & rising clk edge)

bool hli
Using high level interrupt or not

uint32_t magic
Magic number

struct esp_vhci_host_callback

`esp_vhci_host_callback` used for vhci call host function to notify what host need to do

**Public Members**
void (*notify_host_send_available)(void)
    callback used to notify that the host can send packet to controller

int (*notify_host_recv)(uint8_t *data, uint16_t len)
    callback used to notify that the controller has a packet to send to the host

**Macros**

`ESP_BT_CONTROLLER_CONFIG_MAGIC_VAL`  
`BT_CONTROLLER_INIT_CONFIG_DEFAULT()`

**Type Definitions**

typedef struct `esp_vhci_host_callback` `esp_vhci_host_callback_t`  

`esp_vhci_host_callback` used for vhci call host function to notify what host need to do

**Enumerations**

enum `esp_bt_mode_t`  
    Bluetooth mode for controller enable/disable.  
    Values:

    enumerator `ESP_BT_MODE_IDLE`  
        Bluetooth is not running

    enumerator `ESP_BT_MODE_BLE`  
        Run BLE mode

    enumerator `ESP_BT_MODE_CLASSIC_BT`  
        Run Classic BT mode

    enumerator `ESP_BT_MODE_BTDM`  
        Run dual mode

enum [anonymous]  
    BLE sleep clock accuracy(SCA), values for ble_sca field in `esp_bt_controller_config_t`, currently only `ESP_BLE_SCA_500PPM` and `ESP_BLE_SCA_250PPM` are supported.  
    Values:

    enumerator `ESP_BLE_SCA_500PPM`  
        BLE SCA at 500ppm

    enumerator `ESP_BLE_SCA_250PPM`  
        BLE SCA at 250ppm

    enumerator `ESP_BLE_SCA_150PPM`  
        BLE SCA at 150ppm
enum `ESP_BLE_SCA_100PPM`
BLE SCA at 100ppm

enum `ESP_BLE_SCA_75PPM`
BLE SCA at 75ppm

enum `ESP_BLE_SCA_50PPM`
BLE SCA at 50ppm

enum `ESP_BLE_SCA_30PPM`
BLE SCA at 30ppm

enum `ESP_BLE_SCA_20PPM`
BLE SCA at 20ppm

def enum `esp_bt_controller_status_t`
Bluetooth controller enable/disable/initialised/de-initialised status.

Values:

enum `ESP_BT_CONTROLLER_STATUS_IDLE`

enum `ESP_BT_CONTROLLER_STATUS_INITED`

enum `ESP_BT_CONTROLLER_STATUS_ENABLED`

enum `ESP_BT_CONTROLLER_STATUS_NUM`

enum `esp_ble_power_type_t`
BLE tx power type ESP_BLE_PWR_TYPE_CONN_HDL0-8: for each connection, and only be set after connection completed. when disconnect, the correspond TX power is not effected.


ESP_BLE_PWR_TYPE_DEFAULT : if each connection’s TX power is not set, it will use this default value.

if neither in scan mode nor in adv mode, it will use this default value. If none of power type is set, system will use ESP_PWR_LVL_P3 as default for ADV/SCAN/CONN0-9.

Values:

enum `ESP_BLE_PWR_TYPE_CONN_HDL0`
For connection handle 0

enum `ESP_BLE_PWR_TYPE_CONN_HDL1`
For connection handle 1

enum `ESP_BLE_PWR_TYPE_CONN_HDL2`
For connection handle 2

enum `ESP_BLE_PWR_TYPE_CONN_HDL3`
For connection handle 3
enumerator \texttt{ESP_BLE_PWR_TYPE_CONN_HDL4} \\
For connection handle 4

enumerator \texttt{ESP_BLE_PWR_TYPE_CONN_HDL5} \\
For connection handle 5

enumerator \texttt{ESP_BLE_PWR_TYPE_CONN_HDL6} \\
For connection handle 6

enumerator \texttt{ESP_BLE_PWR_TYPE_CONN_HDL7} \\
For connection handle 7

enumerator \texttt{ESP_BLE_PWR_TYPE_CONN_HDL8} \\
For connection handle 8

enumerator \texttt{ESP_BLE_PWR_TYPE_ADV} \\
For advertising

enumerator \texttt{ESP_BLE_PWR_TYPE_SCAN} \\
For scan

enumerator \texttt{ESP_BLE_PWR_TYPE_DEFAULT} \\
For default, if not set other, it will use default value

enumerator \texttt{ESP_BLE_PWR_TYPE_NUM} \\
TYPE numbers

\texttt{enum esp_power_level_t} \\
Bluetooth TX power level(index), it’s just a index corresponding to power(dbm).

\texttt{Values:}

enumerator \texttt{ESP_PWR_LVL_N12} \\
Corresponding to -12dbm

enumerator \texttt{ESP_PWR_LVL_N9} \\
Corresponding to -9dbm

enumerator \texttt{ESP_PWR_LVL_N6} \\
Corresponding to -6dbm

enumerator \texttt{ESP_PWR_LVL_N3} \\
Corresponding to -3dbm

enumerator \texttt{ESP_PWR_LVL_N0} \\
Corresponding to 0dbm

enumerator \texttt{ESP_PWR_LVL_P3} \\
Corresponding to +3dbm
Chapter 2. API 參考

enumerator ESP_PWR_LVL_P6
Corresponding to +6dbm

enumerator ESP_PWR_LVL_P9
Corresponding to +9dbm

enumerator ESP_PWR_LVL_N14
Backward compatibility! Setting to -14dbm will actually result to -12dbm

enumerator ESP_PWR_LVL_N11
Backward compatibility! Setting to -11dbm will actually result to -9dbm

enumerator ESP_PWR_LVL_N8
Backward compatibility! Setting to -8dbm will actually result to -6dbm

enumerator ESP_PWR_LVL_N5
Backward compatibility! Setting to -5dbm will actually result to -3dbm

enumerator ESP_PWR_LVL_N2
Backward compatibility! Setting to -2dbm will actually result to 0dbm

enumerator ESP_PWR_LVL_P1
Backward compatibility! Setting to +1dbm will actually result to +3dbm

enumerator ESP_PWR_LVL_P4
Backward compatibility! Setting to +4dbm will actually result to +6dbm

enumerator ESP_PWR_LVL_P7
Backward compatibility! Setting to +7dbm will actually result to +9dbm

enum esp_sco_data_path_t
Bluetooth audio data transport path.

Values:

enumerator ESP_SCO_DATA_PATH_HCI
data over HCI transport

enumerator ESP_SCO_DATA_PATH_PCM
data over PCM interface

2.3.5 ESP-BLE-MESH

With various features of ESP-BLE-MESH, users can create a managed flooding mesh network for several scenarios, such as lighting, sensor and etc.

For an ESP32 to join and work on a ESP-BLE-MESH network, it must be provisioned firstly. By provisioning, the ESP32, as an un provisioned device, will join the ESP-BLE-MESH network and become a ESP-BLE-MESH node, communicating with other nodes within or beyond the radio range.
Apart from ESP-BLE-MESH nodes, inside ESP-BLE-MESH network, there is also ESP32 that works as ESP-BLE-MESH Provisioner, which could provision unprovisioned devices into ESP-BLE-MESH nodes and configure the nodes with various features.

For information how to start using ESP32 and ESP-BLE-MESH, please see the Section `ESP-BLE-MESH 快速入门`. If you are interested in information on ESP-BLE-MESH architecture, including some details of software implementation, please see Section `ESP-BLE-MESH 架构`.

Application Examples and Demos

Please refer to Sections `ESP-BLE-MESH 示例` and `ESP-BLE-MESH 演示视频`.

API Reference

ESP-BLE-MESH APIs are divided into the following parts:

- `ESP-BLE-MESH Definitions`
- `ESP-BLE-MESH Core API Reference`
- `ESP-BLE-MESH Models API Reference`

ESP-BLE-MESH Definitions

This section contains only one header file, which lists the following items of ESP-BLE-MESH.

- ID of all the models and related message opcodes
- Structs of model, element and Composition Data
- Structs of used by ESP-BLE-MESH Node/Provisioner for provisioning
- Structs used to transmit/receive messages
- Event types and related event parameters

Header File

- `components/bt/esp_ble_mesh/api/esp_ble_mesh_defs.h`

Unions

union `esp_ble_mesh_prov_cb_param_t`

```c
#include <esp_ble_mesh_defs.h> BLE Mesh Node/Provisioner callback parameters union.
```

Public Members

```c
struct esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_register_comp_param prov_register_comp
    Event parameter of ESP_BLE_MESH_PROV_REGISTER_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_set_unprov_dev_name_comp_param
    node_set_unprov_dev_name_comp
    Event parameter of ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_enable_comp_param
    node_prov_enable_comp
    Event parameter of ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT
```
struct esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_disable_comp_param
node_prov_disable_comp
Event parameter of ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_link_open_evt_param
node_prov_link_open
Event parameter of ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_link_close_evt_param
node_prov_link_close
Event parameter of ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_output_num_evt_param
node_prov_output_num
Event parameter of ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_output_str_evt_param
node_prov_output_str
Event parameter of ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_input_evt_param
node_prov_input
Event parameter of ESP_BLE_MESH_NODE_PROV_INPUT_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provision_complete_evt_param
node_prov_complete
Event parameter of ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provision_reset_param
node_prov_reset
Event parameter of ESP_BLE_MESH_NODE_PROV_RESET_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_set_oob_pub_key_comp_param
node_prov_set_oob_pub_key_comp
Event parameter of ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_input_number_comp_param
node_prov_input_numComp
Event parameter of ESP_BLE_MESH_NODE_PROV_INPUT_NUM_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_input_string_comp_param
node_prov_input_str_comp
Event parameter of ESP_BLE_MESH_NODE_PROV_INPUT_STR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_identity_enable_comp_param
node_proxy_identity_enable_comp
Event parameter of ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_gatt_enable_comp_param
node_proxy_gatt_enable_comp
Event parameter of ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_gatt_disable_comp_param
node_proxy_gatt_disable_comp
Event parameter of ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT
struct esp_ble_mesh_prov_cb_param_t::ble_mesh_node_add_local_net_key_comp_param
node_add_net_key_comp
  Event parameter of ESP_BLE_MESH_NODE_ADD_LOCAL_NET_KEY_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_node_add_local_app_key_comp_param
node_add_app_key_comp
  Event parameter of ESP_BLE_MESH_NODE_ADD_LOCAL_APP_KEY_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_node_bind_local_mod_app_comp_param
node_bind_app_key_to_model_comp
  Event parameter of ESP_BLE_MESH_NODE_BIND_APP_KEY_TO_MODEL_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param
provisioner_recv_unprov_adv_pkt
  Event parameter of ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_enable_comp_param
provisioner_prov_enable_comp
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_disable_comp_param
provisioner_prov_disable_comp
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_open_evt_param
provisioner_prov_link_open
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisionerProv_read_oob_pub_key_evt_param
provisioner_prov_read_oob_pub_key
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_input_evt_param
provisioner_prov_input
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_output_evt_param
provisioner_prov_output
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_link_close_evt_param
provisioner_prov_link_close
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete
provisioner_prov_complete
  Event parameter of ESP_BLE_MESH_PROVISIONER_PROV COMPLETE_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_add_unprov_dev_comp_param
provisioner_add_unprov_dev_comp
  Event parameter of ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT
struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_dev_with_addr_comp_param
provisioner_prov_dev_with_addr_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_comp_param
provisioner_delete_dev_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_dev_uuid_match_comp_param
provisioner_set_dev_uuid_match_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_dev_data_info_comp_param
provisioner_set_dev_data_info_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_SET_DEV_DATA_INFO_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_static_oob_val_comp_param
provisioner_set_static_oob_val_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_primary_elem_addr_comp_param
provisioner_set_primary_elem_addr_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_read_oob_pub_key_comp_param
provisioner_prov_read_oob_pub_key_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_input_num_comp_param
provisioner_prov_input_num_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_input_str_comp_param
provisioner_prov_input_str_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_node_name_comp_param
provisioner_set_node_name_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_add_local_app_key_comp_param
provisioner_add_app_key_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_update_local_app_key_comp_param
provisioner_update_app_key_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_bind_local_mod_app_comp_param
provisioner_bind_app_key_to_model_comp
Event parameter of ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT
struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_add_local_net_key_comp_param
definition
This is the event parameter for ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_EVT.

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_update_local_net_key_comp_param
definition
This is the event parameter for ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT.

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_store_node_comp_data_comp_param
definition
This is the event parameter for ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT.

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_node_with_uuid_comp_param
definition
This is the event parameter for ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_COMP_EVT.

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_node_with_addr_comp_param
definition
This is the event parameter for ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT.

int err_code
This indicates the result of enabling/disabling to receive heartbeat messages by the Provisioner.
This indicates the result of setting the heartbeat filter type by the Provisioner.
This indicates the result of setting the heartbeat filter address by the Provisioner.
This indicates the result of directly erasing settings by the Provisioner.
This indicates the result of opening settings with index by the Provisioner.
This indicates the result of opening settings with user id by the Provisioner.
This indicates the result of closing settings with index by the Provisioner.
This indicates the result of closing settings with user id by the Provisioner.
This indicates the result of deleting settings with index by the Provisioner.
This indicates the result of deleting settings with user id by the Provisioner.

bool enable
This indicates enabling or disabling receiving heartbeat messages.

struct esp_ble_mesh_prov_cb_param_t::[anonymous]
definition
This is the event parameter for ESP_BLE_MESH_PROVISIONER_ENABLE_HEARTBEAT_RECV_COMP_EVT.

uint8_t type
This is the type of the filter used for receiving heartbeat messages.

struct esp_ble_mesh_prov_cb_param_t::[anonymous]
definition
This is the event parameter for ESP_BLE_MESH_PROVISIONER_SET_HEARTBEAT_FILTER_TYPE_COMP_EVT.
uint8_t \textbf{op}
Operation (add, remove, clean)

uint16_t \textbf{hb_src}
Heartbeat source address

uint16_t \textbf{hb_dst}
Heartbeat destination address

\textbf{struct esp_ble_mesh_prov_cb_param_t::{[anonymous]}}
\textbf{provisioner_set_heartbeat_filter_info_comp}
ESP_BLE_MESH_PROVISIONER_SET_HEARTBEAT_FILTER_INFO_COMP_EVT.
Event parameters of ESP_BLE_MESH_PROVISIONER_SET_HEARTBEAT_FILTER_INFO_COMP_EVT

uint8_t \textbf{init_ttl}
Heartbeat InitTTL

uint8_t \textbf{rx_ttl}
Heartbeat RxTTL

uint8_t \textbf{hops}
Heartbeat hops (InitTTL - RxTTL + 1)

uint16_t \textbf{feature}
Bit field of currently active features of the node

int8_t \textbf{rssi}
RSSI of the heartbeat message

\textbf{struct esp_ble_mesh_prov_cb_param_t::{[anonymous]}} \textbf{provisioner_recv_heartbeat}
ESP_BLE_MESH_PROVISIONER_RECV_HEARTBEAT_MESSAGE_EVT.
Event parameters of ESP_BLE_MESH_PROVISIONER_RECV_HEARTBEAT_MESSAGE_EVT

\textbf{struct esp_ble_mesh_prov_cb_param_t::{[anonymous]}} \textbf{provisioner_direct_erase_settings_comp}
ESP_BLE_MESH_PROVISIONER_DIRECT_ERASE_SETTINGS_COMP_EVT.
Event parameters of ESP_BLE_MESH_PROVISIONER_DIRECT_ERASE_SETTINGS_COMP_EVT

uint8_t \textbf{index}
Index of Provisioner settings

\textbf{struct esp_ble_mesh_prov_cb_param_t::{[anonymous]}} \textbf{provisioner_open_settings_with_index_comp}
ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_INDEX_COMP_EVT.
Event parameter of ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_INDEX_COMP_EVT

\textbf{char \texttt{uid}[ESP_BLE_MESH_SETTINGS_UID_SIZE + 1]}
Provisioner settings user id
Chapter 2. API

struct esp_ble_mesh_prov_cb_param_t::[anonymous]
provisioner_open_settings_with_uid_comp
ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_UID_COMP_EVT.
Event parameters of ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_UID_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::[anonymous]
provisioner_close_settings_with_index_comp
ESP_BLE_MESH_PROVISIONER_CLOSE_SETTINGS_WITH_INDEX_COMP_EVT.
Event parameter of ESP_BLE_MESH_PROVISIONER_CLOSE_SETTINGS_WITH_INDEX_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::[anonymous]
provisioner_close_settings_with_uid_comp
ESP_BLE_MESH_PROVISIONER_CLOSE_SETTINGS_WITH_UID_COMP_EVT.
Event parameters of ESP_BLE_MESH_PROVISIONER_CLOSE_SETTINGS_WITH_INDEX_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::[anonymous]
provisioner_delete_settings_with_index_comp
ESP_BLE_MESH_PROVISIONER_DELETE_SETTINGS_WITH_INDEX_COMP_EVT.
Event parameter of ESP_BLE_MESH_PROVISIONER_DELETE_SETTINGS_WITH_INDEX_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::[anonymous]
provisioner_delete_settings_with_uid_comp
ESP_BLE_MESH_PROVISIONER_DELETE_SETTINGS_WITH_UID_COMP_EVT.
Event parameters of ESP_BLE_MESH_PROVISIONER_DELETE_SETTINGS_WITH_INDEX_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param
set_fast_prov_info_comp
Event parameter of ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_action_comp_param
set_fast_prov_action_comp
Event parameter of ESP_BLE_MESH_SET_FAST_PROV_ACTION_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_heartbeat_msg_recv_param
heartbeat_msg_recv
Event parameter of ESP_BLE_MESH_HEARTBEAT_MESSAGE_RECV_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_enable_comp_param
lpn_enable_comp
Event parameter of ESP_BLE_MESH_LPN_ENABLE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_disable_comp_param
lpn_disable_comp
Event parameter of ESP_BLE_MESH_LPN_DISABLE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_poll_comp_param
lpn_poll_comp
Event parameter of ESP_BLE_MESH_LPN POLL_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_establish_param
lpn_friendship_establish
Event parameter of ESP_BLE_MESH_LPN FRIENDSHIP_ESTABLISH_EVT
struct esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_terminate_param
lpn_friendship_terminate
Event parameter of ESP_BLE_MESH_LPN_FRIENDSHIP_TERMINATE_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_friend_friendship_establish_param
frnd_friendship_establish
Event parameter of ESP_BLE_MESH_FRIEND_FRIENDSHIP_ESTABLISH_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_friend_friendship_terminate_param
frnd_friendship_terminate
Event parameter of ESP_BLE_MESH_FRIEND_FRIENDSHIP_TERMINATE_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_recv_adv_pkt_param
proxy_client_recv_adv_pkt
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_connected_param
proxy_client_connected
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_CONNECTED_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_disconnected_param
proxy_client_disconnected
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_recv_filter_status_param
proxy_client_recv_filter_status
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_connect_comp_param
proxy_client_connect_comp
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_CONNECT_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_disconnect_comp_param
proxy_client_disconnect_comp
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_set_filter_type_comp_param
proxy_client_set_filter_type_comp
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_add_filter_addr_comp_param
proxy_client_add_filter_addr_comp
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_remove_filter_addr_comp_param
proxy_client_remove_filter_addr_comp
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_server_connected_param
proxy_server_connected
Event parameter of ESP_BLE_MESH_PROXY_SERVER_CONNECTED_EVT
struct esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_server_disconnected_param

proxy_server_disconnected
Event parameter of ESP_BLE_MESH_PROXY_SERVER_DISCONNECTED_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_model_sub_group_addr_comp_param

model_sub_group_addr_comp
Event parameters of ESP_BLE_MESH_MODEL_SUBSCRIBE_GROUP_ADDR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_model_unsub_group_addr_comp_param

model_unsub_group_addr_comp
Event parameters of ESP_BLE_MESH_MODEL_UNSUBSCRIBE_GROUP_ADDR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_deinit_mesh_comp_param

deinit_mesh_comp
Event parameter of ESP_BLE_MESH_DEINIT_MESH_COMP_EVT

#include <esp_ble_mesh_defs.h>
ESP_BLE_MESH_DEINIT_MESH_COMP_EVT.

Public Members

int err_code
Indicate the result of BLE Mesh deinitialization

struct ble_mesh_friend_friendship_establish_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_FRIEND_FRIENDSHIP_ESTABLISH_EVT.

Public Members

uint16_t lpn_addr
Low Power Node unicast address

struct ble_mesh_friend_friendship_terminate_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_FRIEND_FRIENDSHIP_TERMINATE_EVT.

Public Types

enum [anonymous]
This enum value is the reason of friendship termination on the friend node side

Values:

tenumerator ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_ESTABLISH_FAIL
Friend Offer has been sent, but Friend Offer is not received within 1 second, friendship fails to be established

tenumerator ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_POLL_TIMEOUT
Friendship is established, PollTimeout timer expires and no Friend Poll/Sub Add/Sub Remove is received
enumerator **ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_RECV_FRND_REQ**
Receive Friend Request from existing Low Power Node

enumerator **ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_RECV_FRND_CLEAR**
Receive Friend Clear from other friend node

enumerator **ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_DISABLE**
Friend feature disabled or corresponding NetKey is deleted

**Public Members**

```
uint16_t lpn_addr
    Low Power Node unicast address
```

```
enum esp_ble_mesh_prov_cb_param_t::ble_mesh_friend_friendship_terminate_param::[anonymous]
reason
    This enum value is the reason of friendship termination on the friend node side Friendship terminated
```

```
struct ble_mesh_heartbeat_msg_recv_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_HEARTBEAT_MESSAGE_RECV_EVT.
```

**Public Members**

```
uint8_t hops
    Heartbeat hops (InitTTL - RxTTL + 1)
```

```
uint16_t feature
    Bit field of currently active features of the node
```

```
struct ble_mesh_input_evt_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_INPUT_EVT.
```

**Public Members**

```
    esp_ble_mesh_input_action_t action
    Action of Input OOB Authentication
```

```
uint8_t size
    Size of Input OOB Authentication
```

```
struct ble_mesh_input_number_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_INPUT_NUM_COMP_EVT.
```
Chapter 2. API

Public Members

```c
int err_code
```
Indicate the result of inputting number

```c
struct ble_mesh_input_string_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_INPUT_STR_COMP_EVT.
```

Public Members

```c
int err_code
```
Indicate the result of inputting string

```c
struct ble_mesh_link_close_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT.
```

Public Members

```c
esp_ble_mesh_prov_bearer_t bearer
```
Type of the bearer used when device link is closed

```c
struct ble_mesh_link_open_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT.
```

Public Members

```c
esp_ble_mesh_prov_bearer_t bearer
```
Type of the bearer used when device link is open

```c
struct ble_mesh_lpn_disable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_DISABLE_COMP_EVT.
```

Public Members

```c
int err_code
```
Indicate the result of disabling LPN functionality

```c
struct ble_mesh_lpn_enable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_ENABLE_COMP_EVT.
```

Public Members

```c
int err_code
```
Indicate the result of enabling LPN functionality
**struct ble_mesh_lpn_friendship_establish_param**

```
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_FRIENDSHIP_ESTABLISH_EVT.
```

**Public Members**

- `uint16_t friend_addr`  
  Friend Node unicast address

**struct ble_mesh_lpn_friendship_terminate_param**

```
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_FRIENDSHIP_TERMINATE_EVT.
```

**Public Members**

- `uint16_t friend_addr`  
  Friend Node unicast address

**struct ble_mesh_lpn_poll_comp_param**

```
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_POLL_COMP_EVT.
```

**Public Members**

- `int err_code`  
  Indicate the result of sending Friend Poll

**struct ble_mesh_model_sub_group_addr_comp_param**

```
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_SUBSCRIBE_GROUP_ADDR_COMP_EVT.
```

**Public Members**

- `int err_code`  
  Indicate the result of local model subscribing group address
  - `uint16_t element_addr`  
    Element address
  - `uint16_t company_id`  
    Company ID
  - `uint16_t model_id`  
    Model ID
  - `uint16_t group_addr`  
    Group Address

**struct ble_mesh_model_unsub_group_addr_comp_param**

```
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_UNSUBSCRIBE_GROUP_ADDR_COMP_EVT.
```
Chapter 2. API 参考

**Public Members**

```c
int err_code
   Indicate the result of local model unsubscribing group address
```

```c
uint16_t element_addr
   Element address
```

```c
uint16_t company_id
   Company ID
```

```c
uint16_t model_id
   Model ID
```

```c
uint16_t group_addr
   Group Address
```

```c
struct ble_mesh_node_add_local_app_key_comp_param
   #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_ADD_LOCAL_APP_KEY_COMP_EVT.
```

**Public Members**

```c
int err_code
   Indicate the result of adding local AppKey by the node
```

```c
uint16_t net_idx
   NetKey Index
```

```c
uint16_t app_idx
   AppKey Index
```

```c
struct ble_mesh_node_add_local_net_key_comp_param
   #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_ADD_LOCAL_NET_KEY_COMP_EVT.
```

**Public Members**

```c
int err_code
   Indicate the result of adding local NetKey by the node
```

```c
uint16_t net_idx
   NetKey Index
```

```c
struct ble_mesh_node_bind_local_mod_app_comp_param
   #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_BIND_APP_KEY_TO_MODEL_COMP_EVT.
```
Public Members

int err_code
   Indicate the result of binding AppKey with model by the node

uint16_t element_addr
   Element address

uint16_t app_idx
   AppKey Index

uint16_t company_id
   Company ID

uint16_t model_id
   Model ID

struct ble_mesh_output_num_evt_param
   #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT.

Public Members

esp_ble_mesh_output_action_t action
   Action of Output OOB Authentication

uint32_t number
   Number of Output OOB Authentication

struct ble_mesh_output_str_evt_param
   #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT.

Public Members

char string[8]
   String of Output OOB Authentication

struct ble_mesh_prov_disable_comp_param
   #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT.

Public Members

int err_code
   Indicate the result of disabling BLE Mesh device

struct ble_mesh_prov_enable_comp_param
   #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT.
Public Members

```c
int err_code
```
Indicate the result of enabling BLE Mesh device

```c
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROV_REGISTER_COMP_EVT.
```

Public Members

```c
int err_code
```
Indicate the result of BLE Mesh initialization

```c
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT.
```

Public Members

```c
uint16_t net_idx
```
NetKey Index

```c
uint8_t net_key[16]
```
NetKey

```c
uint16_t addr
```
Primary address

```c
uint8_t flags
```
Flags

```c
uint32_t iv_index
```
IV Index

```c
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_RESET_EVT.
```

```c
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT.
```

Public Members

```c
int err_code
```
Indicate the result of adding local AppKey by the Provisioner

```c
uint16_t net_idx
```
NetKey Index
uint16_t **app_idx**  
AppKey Index

struct `ble_mesh_provisioner_add_local_net_key_comp_param`  
#include `<esp_ble_mesh_defs.h>` ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_EVT.

**Public Members**

- `int err_code`  
  Indicate the result of adding local NetKey by the Provisioner

- `uint16_t net_idx`  
  NetKey Index

struct `ble_mesh_provisioner_add_unprov_dev_comp_param`  
#include `<esp_ble_mesh_defs.h>` ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT.

**Public Members**

- `int err_code`  
  Indicate the result of adding device into queue by the Provisioner

struct `ble_mesh_provisioner_bind_local_mod_app_comp_param`  
#include `<esp_ble_mesh_defs.h>` ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT.

**Public Members**

- `int err_code`  
  Indicate the result of binding AppKey with model by the Provisioner

- `uint16_t element_addr`  
  Element address

- `uint16_t app_idx`  
  AppKey Index

- `uint16_t company_id`  
  Company ID

- `uint16_t model_id`  
  Model ID

struct `ble_mesh_provisioner_delete_dev_comp_param`  
#include `<esp_ble_mesh_defs.h>` ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT.
Public Members

```c
int err_code
Indicates the result of deleting device by the Provisioner
```

```c
struct ble_mesh_provisioner_delete_node_with_addr_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT.
```

Public Members

```c
int err_code
Indicates the result of deleting node with unicast address by the Provisioner
```

```c
uint16_t unicast_addr
Node unicast address
```

```c
struct ble_mesh_provisioner_delete_node_with_uuid_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_COMP_EVT.
```

Public Members

```c
int err_code
Indicates the result of deleting node with uuid by the Provisioner
```

```c
uint8_t uuid[16]
Node device uuid
```

```c
struct ble_mesh_provisioner_link_close_evt_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT.
```

Public Members

```c
esp_ble_mesh_prov_bearer_t bearer
Type of the bearer used when Provisioner link is closed
```

```c
uint8_t reason
Reason of the closed provisioning link
```

```c
struct ble_mesh_provisioner_link_open_evt_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT.
```

Public Members

```c
esp_ble_mesh_prov_bearer_t bearer
Type of the bearer used when Provisioner link is opened
```
struct ble_mesh_provisioner_prov_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT.

Public Members

uint16_t node_idx
Index of the provisioned device

esp_ble_mesh_octet16_t device_uuid
Device UUID of the provisioned device

uint16_t unicast_addr
Primary address of the provisioned device

uint8_t element_num
Element count of the provisioned device

uint16_t netkey_idx
NetKey Index of the provisioned device

struct ble_mesh_provisioner_prov_dev_with_addr_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP_EVT.

Public Members

int err_code
Indicate the result of Provisioner starting to provision a device

struct ble_mesh_provisioner_prov_disable_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT.

Public Members

int err_code
Indicate the result of disabling BLE Mesh Provisioner

struct ble_mesh_provisioner_prov_enable_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT.

Public Members

int err_code
Indicate the result of enabling BLE Mesh Provisioner

struct ble_mesh_provisioner_prov_input_evt_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT.
**Public Members**

`esp_ble_mesh_oob_method_t method`
Method of device Output OOB Authentication

`esp_ble_mesh_output_action_t action`
Action of device Output OOB Authentication

`uint8_t size`
Size of device Output OOB Authentication

`uint8_t link_idx`
Index of the provisioning link

```c
#include<esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_EVT.
```

**Public Members**

`int err_code`
Indicate the result of inputting number by the Provisioner

```c
#include<esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EVT.
```

**Public Members**

`int err_code`
Indicate the result of inputting string by the Provisioner

```c
#include<esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT.
```

**Public Members**

`esp_ble_mesh_oob_method_t method`
Method of device Input OOB Authentication

`esp_ble_mesh_input_action_t action`
Action of device Input OOB Authentication

`uint8_t size`
Size of device Input OOB Authentication

`uint8_t link_idx`
Index of the provisioning link

```c
#include<esp_ble_mesh_defs.h>
```
char string[8]
String output by the Provisioner

uint32_t number
Number output by the Provisioner

union esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_output_evt_param::[anonymous]
[anonymous]

struct ble_mesh_provisioner_prov_read_oob_pub_key_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB Púb_KEY_COMP_EVT.

Public Members

int err_code
Indicate the result of setting OOB Public Key by the Provisioner

struct ble_mesh_provisioner_prov_read_oob_pub_key_evt_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB Púb_KEY_EVT.

Public Members

uint8_t link_idx
Index of the provisioning link

struct ble_mesh_provisioner_recv_unprov_adv_pkt_param
#include <esp_ble mesh defs.h> ESP_BLE_MESH_PROVISIONER_RECV_UNPROV ADV_PKT_EVT.

Public Members

uint8_t dev_uuid[16]
Device UUID of the unprovisioned device

esp_ble_mesh_bd_addr_t addr
Device address of the unprovisioned device

esp_ble_mesh_addr_type_t addr_type
Device address type

uint16_t oob_info
OOB Info of the unprovisioned device

uint8_t adv_type
Advertising type of the unprovisioned device


`esp_ble_mesh_prov_bearer`  
Bearer of the unprovisioned device

`int8_t rssi`  
RSSI of the received advertising packet

`struct ble_mesh_provisioner_set_dev_uuid_match_comp_param`

```c
#include <esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT.
```

### Public Members

- `int err_code`
  - Indicate the result of setting Device UUID match value by the Provisioner

`struct ble_mesh_provisioner_set_node_name_comp_param`

```c
#include <esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER.Set.NODE_NAME_COMP_EVT.
```

### Public Members

- `int err_code`
  - Indicate the result of setting provisioned device name by the Provisioner

- `uint16_t node_index`
  - Index of the provisioned device

`struct ble_mesh_provisioner_set_primary_elem_addr_comp_param`

```c
#include <esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT.
```

### Public Members

- `int err_code`
  - Indicate the result of setting unicast address of primary element by the Provisioner

`struct ble_mesh_provisioner_set_prov_data_info_comp_param`

```c
#include <esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT.
```

### Public Members

- `int err_code`
  - Indicate the result of setting provisioning info by the Provisioner

`struct ble_mesh_provisioner_set_static_oob_val_comp_param`

```c
#include <esp_ble_mesh_defs.h>
ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT.
```
Public Members

int err_code
Indicate the result of setting static oob value by the Provisioner

struct ble_mesh_provisioner_store_node_comp_data_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT.

Public Members

int err_code
Indicate the result of storing node composition data by the Provisioner

uint16_t addr
Node element address

struct ble_mesh_provisioner_update_local_app_key_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT.

Public Members

int err_code
Indicate the result of updating local AppKey by the Provisioner

uint16_t net_idx
NetKey Index

uint16_t app_idx
AppKey Index

struct ble_mesh_provisioner_update_local_net_key_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT.

Public Members

int err_code
Indicate the result of updating local NetKey by the Provisioner

uint16_t net_idx
NetKey Index

struct ble_mesh_proxy_client_add_filter_addr_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EVT.
Public Members

int **err_code**
Indicate the result of Proxy Client add filter address

uint8_t **conn_handle**
Proxy connection handle

uint16_t **net_idx**
Corresponding NetKey Index

struct **ble_mesh_proxy_client_connect_comp_param**
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_CONNECT_COMP_EVT.

Public Members

int **err_code**
Indicate the result of Proxy Client connect

*esp_ble_mesh_bd_addr_t* **addr**
Device address of the Proxy Server

*esp_ble_mesh_addr_type_t* **addr_type**
Device address type

uint16_t **net_idx**
Corresponding NetKey Index

struct **ble_mesh_proxy_client_connected_param**
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_CONNECTED_EVT.

Public Members

*esp_ble_mesh_bd_addr_t* **addr**
Device address of the Proxy Server

*esp_ble_mesh_addr_type_t* **addr_type**
Device address type

uint8_t **conn_handle**
Proxy connection handle

uint16_t **net_idx**
Corresponding NetKey Index

struct **ble_mesh_proxy_client_disconnect_comp_param**
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT.
### Public Members

**int** `err_code`

Indicate the result of Proxy Client disconnect

**uint8_t** `conn_handle`

Proxy connection handle

**struct** `ble_mesh_proxy_client_disconnected_param`

#include `<esp_ble_mesh_defs.h>` ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT.

### Public Members

**esp_ble_mesh_bd_addr_t** `addr`

Device address of the Proxy Server

**esp_ble_mesh_addr_type_t** `addr_type`

Device address type

**uint8_t** `conn_handle`

Proxy connection handle

**uint8_t** `reason`

Proxy disconnect reason

**struct** `ble_mesh_proxy_client_recv_adv_pkt_param`

#include `<esp_ble_mesh_defs.h>` ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT.

### Public Members

**esp_ble_mesh_bd_addr_t** `addr`

Device address

**esp_ble_mesh_addr_type_t** `addr_type`

Device address type

**uint16_t** `net_idx`

Network ID related NetKey Index

**uint8_t** `net_id[8]`

Network ID contained in the advertising packet

**int8_t** `rssi`

RSSI of the received advertising packet
struct ble_mesh_proxy_client_recv_filter_status_param
#include <esp_ble_mesh_defs.h>ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT.

Public Members

uint8_t conn_handle
Proxy connection handle

uint16_t server_addr
Proxy Server primary element address

uint16_t net_idx
Corresponding NetKey Index

uint8_t filter_type
Proxy Server filter type (whitelist or blacklist)

uint16_t list_size
Number of addresses in the Proxy Server filter list

struct ble_mesh_proxy_client_remove_filter_addr_comp_param
#include <esp_ble_mesh_defs.h>ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT.

Public Members

int err_code
Indicate the result of Proxy Client remove filter address

uint8_t conn_handle
Proxy connection handle

uint16_t net_idx
Corresponding NetKey Index

struct ble_mesh_proxy_client_set_filter_type_comp_param
#include <esp_ble_mesh_defs.h>ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EVT.

Public Members

int err_code
Indicate the result of Proxy Client set filter type

uint8_t conn_handle
Proxy connection handle
uint16_t net_idx
    Corresponding NetKey Index

struct ble_mesh_proxy_gatt_disable_comp_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT.

Public Members

int err_code
    Indicate the result of disabling Mesh Proxy Service

struct ble_mesh_proxy_gatt_enable_comp_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT.

Public Members

int err_code
    Indicate the result of enabling Mesh Proxy Service

struct ble_mesh_proxy_identity_enable_comp_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT.

Public Members

int err_code
    Indicate the result of enabling Mesh Proxy advertising

struct ble_mesh_proxy_server_connected_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_SERVER_CONNECTED_EVT.

Public Members

uint8_t conn_handle
    Proxy connection handle

struct ble_mesh_proxy_server_disconnected_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_SERVER_DISCONNECTED_EVT.

Public Members

uint8_t conn_handle
    Proxy connection handle

uint8_t reason
    Proxy disconnect reason
struct ble_mesh_set_fast_prov_action_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_SET_FAST_PROV_ACTION_COMP_EVT.

Public Members

uint8_t status_action
Indicate the result of setting action of fast provisioning

struct ble_mesh_set_fast_prov_info_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT.

Public Members

uint8_t status_unicast
Indicate the result of setting unicast address range of fast provisioning

uint8_t status_net_idx
Indicate the result of setting NetKey Index of fast provisioning

uint8_t status_match
Indicate the result of setting matching Device UUID of fast provisioning

struct ble_mesh_set_oob_pub_key_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT.

Public Members

int err_code
Indicate the result of setting OOB Public Key

struct ble_mesh_set_unprov_dev_name_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT.

Public Members

int err_code
Indicate the result of setting BLE Mesh device name

union esp_ble_mesh_server_state_value_t
#include <esp_ble_mesh_defs.h> Server model state value union.

Public Members
uint8_t onoff
    The value of the Generic OnOff state
    The value of the Light LC Light OnOff state

struct esp_ble_mesh_server_state_value_t::[anonymous] gen_onoff
    The Generic OnOff state

int16_t level
    The value of the Generic Level state

struct esp_ble_mesh_server_state_value_t::[anonymous] gen_level
    The Generic Level state

uint8_t onpowerup
    The value of the Generic OnPowerUp state

struct esp_ble_mesh_server_state_value_t::[anonymous] gen_onpowerup
    The Generic OnPowerUp state

uint16_t power
    The value of the Generic Power Actual state

struct esp_ble_mesh_server_state_value_t::[anonymous] gen_power_actual
    The Generic Power Actual state

uint16_t lightness
    The value of the Light Lightness Actual state
    The value of the Light Lightness Linear state
    The value of the Light CTL Lightness state
    The value of the Light HSL Lightness state
    The value of the Light xyl Lightness state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_lightness_actual
    The Light Lightness Actual state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_lightness_linear
    The Light Lightness Linear state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_ctl_lightness
    The Light CTL Lightness state

uint16_t temperature
    The value of the Light CTL Temperature state

int16_t delta_uv
    The value of the Light CTL Delta UV state
struct esp_ble_mesh_server_state_value_t::[anonymous] light_ctl_temp_delta_uv
The Light CTL Temperature & Delta UV states

uint16_t hue
The value of the Light HSL Hue state

uint16_t saturation
The value of the Light HSL Saturation state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_hsl
The Light HSL composite state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_hsl_lightness
The Light HSL Lightness state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_hsl_hue
The Light HSL Hue state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_hsl_saturation
The Light HSL Saturation state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_xyl_lightness
The Light xyL Lightness state

struct esp_ble_mesh_server_state_value_t::[anonymous] light_lc_light_onoff
The Light LC Light OnOff state

union esp_ble_mesh_model_cb_param_t
#include <esp_ble_mesh_defs.h> BLE Mesh model callback parameters union.

Public Members

struct esp_ble_mesh_model_cb_param_t::ble_mesh_model_operation_EVT_param model_operation
Event parameter of ESP_BLE_MESH_MODEL_OPERATION_EVT

struct esp_ble_mesh_model_cb_param_t::ble_mesh_model_send_comp_param model_send_comp
Event parameter of ESP_BLE_MESH_MODEL_SEND_COMP_EVT

struct esp_ble_mesh_model_cb_param_t::ble_mesh_model_publish_comp_param model_publish_comp
Event parameter of ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT

struct esp_ble_mesh_model_cb_param_t::ble_mesh_mod_recv_publish_msg_param client_recv_publish_msg
Event parameter of ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT

struct esp_ble_mesh_model_cb_param_t::ble_mesh_client_model_send_timeout_param client_send_timeout
Event parameter of ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT
struct esp_ble_mesh_model_cb_param_t::ble_mesh_model_publish_update_evt_param

model_publish_update

Event parameter of ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT

struct esp_ble_mesh_model_cb_param_t::ble_mesh_server_model_update_state_comp_param

server_model_update_state

Event parameter of ESP_BLE_MESH_SERVER_MODEL_UPDATE_STATE_COMP_EVT

struct ble_mesh_client_model_send_timeout_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT.

Public Members

uint32_t opcode

Opcode of the previously sent message

esp_ble_mesh_model_t *model

Pointer to the model which sends the previous message

esp_ble_mesh_msg_ctx_t *ctx

Pointer to the context of the previous message

struct ble_mesh_mod_recv_publish_msg_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT.

Public Members

uint32_t opcode

Opcode of the unsolicited received message

esp_ble_mesh_model_t *model

Pointer to the model which receives the message

esp_ble_mesh_msg_ctx_t *ctx

Pointer to the context of the message

uint16_t length

Length of the received message

uint8_t *msg

Value of the received message

struct ble_mesh_model_operation_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_OPERATION_EVT.
Public Members

uint32_t opcode
    Opcode of the received message

esp_ble_mesh_model_t *model
    Pointer to the model which receives the message

esp_ble_mesh_msg_ctx_t *ctx
    Pointer to the context of the received message

uint16_t length
    Length of the received message

uint8_t *msg
    Value of the received message

struct ble_mesh_model_publish_comp_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT.

Public Members

int err_code
    Indicate the result of publishing a message

esp_ble_mesh_model_t *model
    Pointer to the model which publishes the message

struct ble_mesh_model_publish_update_evt_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT.

Public Members

esp_ble_mesh_model_t *model
    Pointer to the model which is going to update its publish message

struct ble_mesh_model_send_comp_param
    #include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_SEND_COMP_EVT.

Public Members

int err_code
    Indicate the result of sending a message

uint32_t opcode
    Opcode of the message
```c
#include<esp_ble_mesh_defs.h>
```

# ESP_BLE_MESH_SERVER_MODEL_UPDATE_STATE_COMP_EVT

## Public Members

- **err_code**
  - `int`  
  - Indicate the result of updating server model state

- **model**
  - `esp_ble_mesh_model_t *`  
  - Pointer to the server model which state value is updated

- **type**
  - `esp_ble_mesh_server_state_type_t`  
  - Type of the updated server state

## Structures

- **esp_ble_mesh_deinit_param_t**
  - BLE Mesh deinit parameters

  - **erase_flash**
    - `bool`  
    - Indicate if erasing flash when deinit mesh stack

- **esp_ble_mesh_elem_t**
  - Abstraction that describes a BLE Mesh Element. This structure is associated with struct bt_mesh_elem in `mesh_access.h`

  - **element_addr**
    - uint16_t  
    - Element Address, assigned during provisioning.

  - **location**
    - const uint16_t  
    - Location Descriptor (GATT Bluetooth Namespace Descriptors)

  - **sig_model_count**
    - const uint8_t  
    - SIG Model count

  - **vnd_model_count**
    - const uint8_t  
    - Vendor Model count

---

*Espresif Systems* 526  
Release v5.1-dev-2066-g7869f4e151  
Submit Document Feedback
**struct esp_ble_mesh_model_pub_t**

Abstraction that describes a model publication context. This structure is associated with struct bt_mesh_model_pub in mesh_access.h

**Public Members**

- `esp_ble_mesh_model_t *model`
  Pointer to the model to which the context belongs. Initialized by the stack.

- `uint16_t publish_addr`
  Publish Address.

- `uint16_t app_idx`
  Publish AppKey Index.

- `uint16_t cred`
  Friendship Credentials Flag.

- `uint16_t send_rel`
  Force reliable sending (segment acks)

- `uint8_t ttl`
  Publish Time to Live.

- `uint8_t retransmit`
  Retransmit Count & Interval Steps.

- `uint8_t period`
  Publish Period.

- `uint8_t period_div`
  Divisor for the Period.

- `uint8_t fast_period`
  Use FastPeriodDivisor

- `uint8_t count`
  Retransmissions left.

- `uint32_t period_start`
  Start of the current period.
struct net_buf_simple *msg
   Publication buffer, containing the publication message.
   This will get correctly created when the publication context has been defined using the
   ESP_BLE_MESH_MODEL_PUB_DEFINE macro.
   ESP_BLE_MESH_MODEL_PUB_DEFINE(name, size);

   esp_ble_mesh_cb_t update
   Callback used to update publish message. Initialized by the stack.

struct k_delayed_work timer
   Publish Period Timer. Initialized by the stack.

uint8_t dev_role
   Role of the device that is going to publish messages

struct esp_ble_mesh_model_op_t
   Abstraction that describes a model operation context. This structure is associated with struct
   bt_mesh_model_op in mesh_access.h

Public Members

const uint32_t opcode
   Message opcode

const size_t min_len
   Message minimum length

   esp_ble_mesh_cb_t param_cb
   Callback used to handle message. Initialized by the stack.

struct esp_ble_mesh_model_cbs_t
   Abstraction that describes a model callback structure. This structure is associated with struct
   bt_mesh_model_cb in mesh_access.h.

Public Members

   esp_ble_mesh_cb_t init_cb
   Callback used during model initialization. Initialized by the stack.

struct esp_ble_mesh_model
   Abstraction that describes a Mesh Model instance. This structure is associated with struct bt_mesh_model in
   mesh_access.h

Public Members
const uint16_t model_id
16-bit model identifier

uint16_t company_id
16-bit company identifier

uint16_t model_id
16-bit model identifier

struct esp_ble_mesh_model::[anonymous]::[anonymous] vnd
Structure encapsulating a model ID with a company ID

union esp_ble_mesh_model::[anonymous] [anonymous] Model ID

union element_idx
Internal information, mainly for persistent storage Belongs to Nth element

uint8_t model_idx
Is the Nth model in the element

tuint16_t flags
Information about what has changed

esp_ble_mesh_elem_t *element
The Element to which this Model belongs

esp_ble_mesh_model_pub_t *const pub
Model Publication

uint16_t keys[CONFIG_BLE_MESH_MODEL_KEY_COUNT]
AppKey List

uint16_t groups[CONFIG_BLE_MESH_MODEL_GROUP_COUNT]
Subscription List (group or virtual addresses)

esp_ble_mesh_model_op_t *op
Model operation context

esp_ble_mesh_model_cbs_t *cb
Model callback structure

void *user_data
Model-specific user data

struct esp_ble_mesh_msg_ctx_t
Messages sending context. This structure is associated with struct bt_mesh_msg_ctx in mesh_access.h
Public Members

uint16_t net_idx
NetKey Index of the subnet through which to send the message.

uint16_t app_idx
AppKey Index for message encryption.

uint16_t addr
Remote address.

uint16_t recv_dst
Destination address of a received message. Not used for sending.

int8_t recv_rssi
RSSI of received packet. Not used for sending.

uint8_t recv_ttl
Received TTL value. Not used for sending.

uint8_t send_rel
Force sending reliably by using segment acknowledgement

uint8_t send_ttl
TTL, or ESP_BLE_MESH_TTL_DEFAULT for default TTL.

uint32_t recv_op
Opcode of a received message. Not used for sending message.

esp_ble_mesh_model_t *model
Model corresponding to the message, no need to be initialized before sending message

bool srv_send
Indicate if the message is sent by a node server model, no need to be initialized before sending message

struct esp_ble_mesh_prov_t
Provisioning properties & capabilities. This structure is associated with struct bt_mesh_prov in mesh_access.h

struct esp_ble_mesh_comp_t
Node Composition data context. This structure is associated with struct bt_mesh_comp in mesh_access.h

Public Members

uint16_t cid
16-bit SIG-assigned company identifier

uint16_t pid
16-bit vendor-assigned product identifier
Chapter 2. API 参考

uint16_t vid
   16-bit vendor-assigned product version identifier

size_t element_count
   Element count

esp_ble_mesh_elem_t *elements
   A sequence of elements

struct esp_ble_mesh_unprov_dev_add_t
   Information of the device which is going to be added for provisioning.

   Public Members

   esp_ble_mesh_bd_addr_t addr
      Device address

   esp_ble_mesh_addr_type_t addr_type
      Device address type

   uint8_t uuid[16]
      Device UUID

   uint16_t oob_info
      Device OOB Info ADD_DEV_START_PROV_NOW_FLAG shall not be set if the bearer has both PB-ADV and PB-GATT enabled

   esp_ble_mesh_prov_bearer_t bearer
      Provisioning Bearer

struct esp_ble_mesh_device_delete_t
   Information of the device which is going to be deleted.

   Public Members

   esp_ble_mesh_bd_addr_t addr
      Device address

   esp_ble_mesh_addr_type_t addr_type
      Device address type

   uint8_t uuid[16]
      Device UUID

   uint8_t flag
      BIT0: device address; BIT1: device UUID
struct `esp_ble_mesh_prov_data_info_t`
Information of the provisioner which is going to be updated.

**Public Members**

`uint16_t net_idx`
NetKey Index

`uint8_t flags`
Flags

`uint32_t iv_index`
IV Index

`uint8_t flag`
BIT0: net_idx; BIT1: flags; BIT2: iv_index

struct `esp_ble_mesh_node_t`
Information of the provisioned node

**Public Members**

`esp_ble_mesh_bd_addr_t addr`
Node device address

`esp_ble_mesh_addr_type_t addr_type`
Node device address type

`uint8_t dev_uuid[16]`
Device UUID

`uint16_t oob_info`
Node OOB information

`uint16_t unicast_addr`
Node unicast address

`uint8_t element_num`
Node element number

`uint16_t net_idx`
Node NetKey Index

`uint8_t flags`
Node key refresh flag and iv update flag
uint32_t iv_index
   Node IV Index

uint8_t dev_key[16]
   Node device key

char name[ESP_BLE_MESH_NODE_NAME_MAX_LEN + 1]
   Node name

uint16_t comp_length
   Length of Composition Data

uint8_t *comp_data
   Value of Composition Data

struct esp_ble_mesh_fast_prov_info_t
   Context of fast provisioning which need to be set.

Public Members

uint16_t unicast_min
   Minimum unicast address used for fast provisioning

uint16_t unicast_max
   Maximum unicast address used for fast provisioning

uint16_t net_idx
   Netkey index used for fast provisioning

uint8_t flags
   Flags used for fast provisioning

uint32_t iv_index
   IV Index used for fast provisioning

uint8_t offset
   Offset of the UUID to be compared

uint8_t match_len
   Length of the UUID to be compared

uint8_t match_val[16]
   Value of UUID to be compared

struct esp_ble_mesh_heartbeat_filter_info_t
   Context of Provisioner heartbeat filter information to be set
Public Members

uint16_t hb_src
Heartbeat source address (unicast address)

uint16_t hb_dst
Heartbeat destination address (unicast address or group address)

struct esp_ble_mesh_client_op_pair_t
BLE Mesh client models related definitions.
Client model Get/Set message opcode and corresponding Status message opcode

Public Members

uint32_t cli_op
The client message opcode

uint32_t status_op
The server status opcode corresponding to the client message opcode

struct esp_ble_mesh_client_t
Client Model user data context.

Public Members

esp_ble_mesh_model_t *model
Pointer to the client model. Initialized by the stack.

int op_pair_size
Size of the op_pair

const esp_ble_mesh_client_op_pair_t *op_pair
Table containing get/set message opcode and corresponding status message opcode

uint32_t publish_status
Callback used to handle the received unsolicited message. Initialized by the stack.

void *internal_data
Pointer to the internal data of client model

uint8_t msg_role
Role of the device (Node/Provisioner) that is going to send messages

struct esp_ble_mesh_client_common_param_t
Common parameters of the messages sent by Client Model.
Chapter 2. API 参考

Public Members

*esp_ble_mesh_opcode_t* **opcode**
Message opcode

*esp_ble_mesh_model_t *model*
Pointer to the client model structure

*esp_ble_mesh_msg_ctx_t ctx*
The context used to send message

int32_t **msg_timeout**
Timeout value (ms) to get response to the sent message. Note: if using default timeout value in menuconfig, make sure to set this value to 0

uint8_t **msg_role**
Role of the device - Node/Provisioner

struct *esp_ble_mesh_state_transition_t*
Parameters of the server model state transition

Public Functions

BLE_MESH_ATOMIC_DEFINE (flag, ESP_BLE_MESH_SERVER_FLAG_MAX)
Flag used to indicate if the transition timer has been started internally.

If the model which contains *esp_ble_mesh_state_transition_t* sets “set_auto_rsp” to ESP_BLE_MESH_SERVER_RSP_BY_APP, the handler of the timer shall be initialized by the users.

And users can use this flag to indicate whether the timer is started or not.

Public Members

bool **just_started**
Indicate if the state transition has just started

uint8_t **trans_time**
State transition time

uint8_t **remain_time**
Remaining time of state transition

uint8_t **delay**
Delay before starting state transition

uint32_t **quo_tt**
Duration of each divided transition step

Submit Document Feedback
**uint32_t counter**  
Number of steps which the transition duration is divided

**uint32_t total_duration**  
State transition total duration

**int64_t start_timestamp**  
Time when the state transition is started

**struct k_delayed_work timer**  
Timer used for state transition

**struct esp_ble_mesh_last_msg_info_t**  
Parameters of the server model received last same set message.

**Public Members**

**uint8_t tid**  
Transaction number of the last message

**uint16_t src**  
Source address of the last message

**uint16_t dst**  
Destination address of the last message

**int64_t timestamp**  
Time when the last message is received

**struct esp_ble_mesh_server_rsp_ctrl_t**  
Parameters of the Server Model response control

**Public Members**

**uint8_t get_auto_rsp**  
BLE Mesh Server Response Option.

i. If `get_auto_rsp` is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, then the response of Client Get messages need to be replied by the application;
ii. If `get_auto_rsp` is set to ESP_BLE_MESH_SERVER_AUTO_RSP, then the response of Client Get messages will be replied by the server models;
iii. If `set_auto_rsp` is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, then the response of Client Set messages need to be replied by the application;
iv. If `set_auto_rsp` is set to ESP_BLE_MESH_SERVER_AUTO_RSP, then the response of Client Set messages will be replied by the server models;
v. If `status_auto_rsp` is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, then the response of Server Status messages need to be replied by the application;
vi. If `status_auto_rsp` is set to ESP_BLE_MESH_SERVER_AUTO_RSP, then the response of Server Status messages will be replied by the server models; Response control for Client Get messages
**uint8_t set_auto_rsp**
Response control for Client Set messages

**uint8_t status_auto_rsp**
Response control for Server Status messages

**Macros**

**ESP_BLE_MESH_SDU_MAX_LEN**
The maximum length of a BLE Mesh message, including Opcode, Payload and TransMIC Length of a short Mesh MIC.

**ESP_BLE_MESH_MIC_SHORT**
Length of a long Mesh MIC.

**ESP_BLE_MESH_MIC_LONG**
The maximum length of a BLE Mesh provisioned node name

**ESP_BLE_MESH_NODE_NAME_MAX_LEN**
The maximum length of a BLE Mesh unprovisioned device name

**ESP_BLE_MESH_DEVICE_NAME_MAX_LEN**
The maximum length of settings user id

**ESP_BLE_MESH_SETTINGS_UID_SIZE**
Invalid settings index

**ESP_BLE_MESH_INVALID_SETTINGS_IDX**
Define the BLE Mesh octet 16 bytes size

**ESP_BLE_MESH_OCTET16_LEN**

**ESP_BLE_MESH_OCTET8_LEN**

**ESP_BLE_MESH_CID_NVAL**
Special TTL value to request using configured default TTL

**ESP_BLE_MESH_TTL_DEFAULT**
Maximum allowed TTL value

**ESP_BLE_MESH_TTL_MAX**

**ESP_BLE_MESH_ADDR_UNASSIGNED**

**ESP_BLE_MESH_ADDR_ALL_NODES**

**ESP_BLE_MESH_ADDR_PROXIES**
ESP_BLE_MESH_ADDR_FRIENDS

ESP_BLE_MESH_ADDR_RELAYS

ESP_BLE_MESH_KEY_UNUSED

ESP_BLE_MESH_KEY_DEV

ESP_BLE_MESH_KEY_PRIMARY

ESP_BLE_MESH_KEY_ANY
   Primary Network Key index

ESP_BLE_MESH_NET_PRIMARY
   Relay state value

ESP_BLE_MESH_RELAY_DISABLED

ESP_BLE_MESH_RELAY_ENABLED

ESP_BLE_MESH_RELAY_NOT_SUPPORTED
   Beacon state value

ESP_BLE_MESH_BEACON_DISABLED

ESP_BLE_MESH_BEACON_ENABLED
   GATT Proxy state value

ESP_BLE_MESH_GATT_PROXY_DISABLED

ESP_BLE_MESH_GATT_PROXY_ENABLED

ESP_BLE_MESH_GATT_PROXY_NOT_SUPPORTED
   Friend state value

ESP_BLE_MESH_FRIEND_DISABLED

ESP_BLE_MESH_FRIEND_ENABLED

ESP_BLE_MESH_FRIEND_NOT_SUPPORTED
   Node identity state value

ESP_BLE_MESH_NODE_IDENTITY_STOPPED

ESP_BLE_MESH_NODE_IDENTITY_RUNNING
Chapter 2. API 参考

ESP_BLE_MESH_NODE_IDENTITY_NOT_SUPPORTED

Supported features

ESP_BLE_MESH_FEATURE_RELAY

ESP_BLE_MESH_FEATURE_PROXY

ESP_BLE_MESH_FEATURE_FRIEND

ESP_BLE_MESH_FEATURE_LOW_POWER

ESP_BLE_MESH_FEATURE_ALL_SUPPORTED

ESP_BLE_MESH_ADDR_IS_UNICAST (addr)

ESP_BLE_MESH_ADDR_IS_GROUP (addr)

ESP_BLE_MESH_ADDR_IS_VIRTUAL (addr)

ESP_BLE_MESH_ADDR_IS_RFU (addr)

ESP_BLE_MESH_INVALID_NODE_INDEX

ESP_BLE_MESH_TRANSMIT (count, int_ms)

Encode transmission count & interval steps.

备注：For example, ESP_BLE_MESH_TRANSMIT(2, 20) means that the message will be sent about 90ms (count is 3, step is 1, interval is 30 ms which includes 10ms of advertising interval random delay).

参数
- count – Number of retransmissions (first transmission is excluded).
- int_ms – Interval steps in milliseconds. Must be greater than 0 and a multiple of 10.

返回 BLE Mesh transmit value that can be used e.g. for the default values of the Configuration Model data.

ESP_BLE_MESH_GET_TRANSMIT_COUNT (transmit)

Decode transmit count from a transmit value.

参数
- transmit – Encoded transmit count & interval value.

返回 Transmission count (actual transmissions equal to N + 1).

ESP_BLE_MESH_GET_TRANSMIT_INTERVAL (transmit)

Decode transmit interval from a transmit value.

参数
- transmit – Encoded transmit count & interval value.

返回 Transmission interval in milliseconds.

ESP_BLE_MESH_PUBLISH_TRANSMIT (count, int_ms)

Encode Publish Retransmit count & interval steps.

参数
- count – Number of retransmissions (first transmission is excluded).
- int_ms – Interval steps in milliseconds. Must be greater than 0 and a multiple of 50.
BLE Mesh transmit value that can be used e.g. for the default values of the Configuration Model data.

**ESP_BLE_MESH_GET_PUBLISH_TRANSMIT_COUNT** (transmit)

Decode Publish Retransmit count from a given value.

参数

- transmit – Encoded Publish Retransmit count & interval value.

返回

Retransmission count (actual transmissions equal to N + 1).

**ESP_BLE_MESH_GET_PUBLISH_TRANSMIT_INTERVAL** (transmit)

Decode Publish Retransmit interval from a given value.

Callbacks which are not needed to be initialized by users (set with 0 and will be initialized internally)

参数

- transmit – Encoded Publish Retransmit count & interval value.

返回

Transmission interval in milliseconds.

**ESP_BLE_MESH_PROV_STATIC_OOB_MAX_LEN**

Maximum length of string used by Output OOB authentication

**ESP_BLE_MESH_PROV_OUTPUT_OOB_MAX_LEN**

Maximum length of string used by Output OOB authentication

**ESP_BLE_MESH_PROV_INPUT_OOB_MAX_LEN**

Macros used to define message opcode

**ESP_BLE_MESH_MODEL_OP_1** (b0)

**ESP_BLE_MESH_MODEL_OP_2** (b0, b1)

**ESP_BLE_MESH_MODEL_OP_3** (b0, cid)

This macro is associated with BLE_MESH_MODEL_CB in mesh_access.h

**ESP_BLE_MESH_SIG_MODEL** (_id, _op, _pub, _user_data)

This macro is associated with BLE_MESH_MODEL_VND_CB in mesh_access.h

**ESP_BLE_MESH_VENDOR_MODEL** (_company, _id, _op, _pub, _user_data)

**ESP_BLE_MESH_ELEMENT** (_loc, _mods, _vnd.mods)

Helper to define a BLE Mesh element within an array.

In case the element has no SIG or Vendor models, the helper macro **ESP_BLE_MESH_MODEL_NONE** can be given instead.

备注: This macro is associated with BLE_MESH_ELEM in mesh_access.h

参数

- _loc – Location Descriptor.
- _mods – Array of SIG models.
- _vnd.mods – Array of vendor models.

**ESP_BLE_MESH_PROV** (uuid, sta_val, sta_val_len, out_size, out_act, in_size, in_act)

**BT_OCTET32_LEN**
BD_ADDR_LEN

ESP_BLE_MESH_ADDR_TYPE_PUBLIC

ESP_BLE_MESH_ADDR_TYPE_RANDOM

ESP_BLE_MESH_ADDR_TYPE_RPA_PUBLIC

ESP_BLE_MESH_ADDR_TYPE_RPA_RANDOM

ESP_BLE_MESH_MODEL_PUB_DEFINE (_name, _msg_len, _role)

Define a model publication context.

参数

• _name – Variable name given to the context.
• _msg_len – Length of the publication message.
• _role – Role of the device which contains the model.

ESP_BLE_MESH_MODEL_OP (_opcode, _min_len)

Define a model operation context.

参数

• _opcode – Message opcode.
• _min_len – Message minimum length.

ESP_BLE_MESH_MODEL_OP_END

Define the terminator for the model operation table. Each model operation struct array must use this terminator as the end tag of the operation unit.

ESP_BLE_MESH_MODEL_NONE

Helper to define an empty model array. This structure is associated with BLE_MESH_MODEL_NONE in mesh_access.h

ADD_DEV_RM_AFTER_PROV_FLAG

Device will be removed from queue after provisioned successfully

ADD_DEV_START_PROV_NOW_FLAG

Start provisioning device immediately

ADD_DEV_FLUSHABLE_DEV_FLAG

Device can be remove when queue is full and new device is going to added

DEL_DEV_ADDR_FLAG

DEL_DEV_UUID_FLAG

PROV_DATA_NET_IDX_FLAG

PROV_DATA_FLAGS_FLAG

PROV_DATA_IV_INDEX_FLAG
ESP_BLE_MESH_HEARTBEAT_FILTER_ACCEPTLIST

ESP_BLE_MESH_HEARTBEAT_FILTER_REJECTLIST
    Provisioner heartbeat filter operation

ESP_BLE_MESH_HEARTBEAT_FILTER_ADD

ESP_BLE_MESH_HEARTBEAT_FILTER_REMOVE

ESP_BLE_MESH_MODEL_ID_CONFIG_SRV
    BLE Mesh models related Model ID and Opcode definitions.
    < Foundation Models

ESP_BLE_MESH_MODEL_ID_CONFIG_CLI

ESP_BLE_MESH_MODEL_ID_HEALTH_SRV

ESP_BLE_MESH_MODEL_ID_HEALTH_CLI
    Models from the Mesh Model Specification

ESP_BLE_MESH_MODEL_ID_GEN_ONOFF_SRV

ESP_BLE_MESH_MODEL_ID_GEN_ONOFF_CLI

ESP_BLE_MESH_MODEL_ID_GEN_LEVEL_SRV

ESP_BLE_MESH_MODEL_ID_GEN_LEVEL_CLI

ESP_BLE_MESH_MODEL_ID_GEN_DEF_TRANS_TIME_SRV

ESP_BLE_MESH_MODEL_ID_GEN_DEF_TRANS_TIME_CLI

ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_CLI

ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_CLI

ESP_BLE_MESH_MODEL_ID_GEN_BATTERY_SRV
Chapter 2. API

ESP_BLE_MESH_MODEL_ID_GEN_BATTERY_CLI
ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_SRV
ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_CLI
ESP_BLE_MESH_MODEL_ID_GEN_ADMIN_PROP_SRV
ESP_BLE_MESH_MODEL_ID_GEN_MANUFACTURER_PROP_SRV
ESP_BLE_MESH_MODEL_ID_GEN_USER_PROP_SRV
ESP_BLE_MESH_MODEL_ID_GEN_CLIENT_PROP_SRV
ESP_BLE_MESH_MODEL_ID_GEN_PROP_CLI
ESP_BLE_MESH_MODEL_ID_SENSOR_SRV
ESP_BLE_MESH_MODEL_ID_SENSOR_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_SENSOR_CLI
ESP_BLE_MESH_MODEL_ID_TIME_SRV
ESP_BLE_MESH_MODEL_ID_TIME_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_TIME_CLI
ESP_BLE_MESH_MODEL_ID_SCENE_SRV
ESP_BLE_MESH_MODEL_ID_SCENE_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_SCENE_CLI
ESP_BLE_MESH_MODEL_ID_SCHEDULER_SRV
ESP_BLE_MESH_MODEL_ID_SCHEDULER_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_SCHEDULER_CLI
ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_SETUP_SRV
Chapter 2. API Reference

ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_CLI
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_CLI
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_TEMP_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_CLI
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_HUE_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SAT_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_CLI
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_CLI
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_HUE_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SAT_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_CLI
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SETUP_SRV
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_CLI
ESP_BLE_MESH_MODEL_OP_BEACON_GET
    Config Beacon Get
ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET
    Config Composition Data Get
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_GET
    Config Default TTL Get
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_GET
    Config GATT Proxy Get
ESP_BLE_MESH_MODEL_OP_RELAY_GET
    Config Relay Get
Chapter 2. API

**ESP_BLE_MESH_MODEL_OP_MODEL_PUB_GET**
Config Model Publication Get

**ESP_BLE_MESH_MODEL_OP_FRIEND_GET**
Config Friend Get

**ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_GET**
Config Heartbeat Publication Get

**ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_GET**
Config Heartbeat Subscription Get

**ESP_BLE_MESH_MODEL_OP_NET_KEY_GET**
Config NetKey Get

**ESP_BLE_MESH_MODEL_OP_APP_KEY_GET**
Config AppKey Get

**ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_GET**
Config Node Identity Get

**ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_GET**
Config SIG Model Subscription Get

**ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_GET**
Config Vendor Model Subscription Get

**ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_GET**
Config SIG Model App Get

**ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_GET**
Config Vendor Model App Get

**ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_GET**
Config Key Refresh Phase Get

**ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_GET**
Config Low Power Node PollTimeout Get

**ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_GET**
Config Network Transmit Get

**ESP_BLE_MESH_MODEL_OP_BEACON_SET**
Config Beacon Set

**ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET**
Config Default TTL Set
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET
Config GATT Proxy Set

ESP_BLE_MESH_MODEL_OP_RELAY_SET
Config Relay Set

ESP_BLE_MESH_MODEL_OP_MODEL_PUB_SET
Config Model Publication Set

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_ADD
Config Model Subscription Add

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_ADD
Config Model Subscription Virtual Address Add

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE
Config Model Subscription Delete

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_DELETE
Config Model Subscription Virtual Address Delete

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_OVERWRITE
Config Model Subscription Overwrite

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_OVERWRITE
Config Model Subscription Virtual Address Overwrite

ESP_BLE_MESH_MODEL_OP_NET_KEY_ADD
Config NetKey Add

ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD
Config AppKey Add

ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND
Config Model App Bind

ESP_BLE_MESH_MODEL_OP_NODE_RESET
Config Node Reset

ESP_BLE_MESH_MODEL_OP_FRIEND_SET
Config Friend Set

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_SET
Config Heartbeat Publication Set

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_SET
Config Heartbeat Subscription Set
ESP_BLE_MESH_MODEL_OP_NET_KEY_UPDATE
  Config NetKey Update

ESP_BLE_MESH_MODEL_OP_NET_KEY_DELETE
  Config NetKey Delete

ESP_BLE_MESH_MODEL_OP_APP_KEY_UPDATE
  Config AppKey Update

ESP_BLE_MESH_MODEL_OP_APP_KEY_DELETE
  Config AppKey Delete

ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_SET
  Config Node Identity Set

ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_SET
  Config Key Refresh Phase Set

ESP_BLE_MESH_MODEL_OP_MODEL_PUB_VIRTUAL_ADDR_SET
  Config Model Publication Virtual Address Set

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE_ALL
  Config Model Subscription Delete All

ESP_BLE_MESH_MODEL_OP_MODEL_APP_UNBIND
  Config Model App Unbind

ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_SET
  Config Network Transmit Set

ESP_BLE_MESH_MODEL_OP_BEACON_STATUS

ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_STATUS

ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_STATUS

ESP_BLE_MESH_MODEL_OP_GATT_PROXY_STATUS

ESP_BLE_MESH_MODEL_OP_RELAY_STATUS

ESP_BLE_MESH_MODEL_OP_MODEL_PUB_STATUS

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_STATUS

ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_LIST

ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_LIST
ESP_BLE_MESH_MODEL_OP_NET_KEY_STATUS
ESP_BLE_MESH_MODEL_OP_NET_KEY_LIST
ESP_BLE_MESH_MODEL_OP_APP_KEY_STATUS
ESP_BLE_MESH_MODEL_OP_APP_KEY_LIST
ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_STATUS
ESP_BLE_MESH_MODEL_OP_MODEL_APP_STATUS
ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_LIST
ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_LIST
ESP_BLE_MESH_MODEL_OP_NODE_RESET_STATUS
ESP_BLE_MESH_MODEL_OP_FRIEND_STATUS
ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_STATUS
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_STATUS
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_STATUS
ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_STATUS
ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_STATUS
ESP_BLE_MESH_CFG_STATUS_SUCCESS
ESP_BLE_MESH_CFG_STATUS_INVALID_ADDRESS
ESP_BLE_MESH_CFG_STATUS_INVALID_MODEL
ESP_BLE_MESH_CFG_STATUS_INVALID_APPKEY
ESP_BLE_MESH_CFG_STATUS_INVALID_NETKEY
ESP_BLE_MESH_CFG_STATUS_INSUFFICIENT_RESOURCES
ESP_BLE_MESH_CFG_STATUS_KEY_INDEX_ALREADY_STORED
ESP_BLE_MESH_CFG_STATUS_INVALID_PUBLISH_PARAMETERS
ESP_BLE_MESH_CFG_STATUS_NOT_A_SUBSCRIBE_MODEL

ESP_BLE_MESH_CFG_STATUS_STORAGE_FAILURE

ESP_BLE_MESH_CFG_STATUS_FEATURE_NOT_SUPPORTED

ESP_BLE_MESH_CFG_STATUS_CANNOT_UPDATE

ESP_BLE_MESH_CFG_STATUS_CANNOT_REMOVE

ESP_BLE_MESH_CFG_STATUS_CANNOT_BIND

ESP_BLE_MESH_CFG_STATUS_TEMP_UNABLE_TO_CHANGE_STATE

ESP_BLE_MESH_CFG_STATUS_CANNOT_SET

ESP_BLE_MESH_CFG_STATUS_UNSPECIFIED_ERROR

ESP_BLE_MESH_CFG_STATUS_INVALID_BINDING

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_GET
  Health Fault Get

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_GET
  Health Period Get

ESP_BLE_MESH_MODEL_OP_ATTENTION_GET
  Health Attention Get

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR
  Health Fault Clear

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR_UNACK
  Health Fault Clear Unacknowledged

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST
  Health Fault Test

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK
  Health Fault Test Unacknowledged

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET
  Health Period Set

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET_UNACK
  Health Period Set Unacknowledged
ESP_BLE_MESH_MODEL_OP_ATTENTION_SET
    Health Attention Set

ESP_BLE_MESH_MODEL_OP_ATTENTION_SET_UNACK
    Health Attention Set Unacknowledged

ESP_BLE_MESH_MODEL_OP_HEALTH_CURRENT_STATUS

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_STATUS

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_STATUS

ESP_BLE_MESH_MODEL_OP_ATTENTION_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_GET

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_STATUS
    Generic Level Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_GET

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET

ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET

ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET_UNACK
    Generic Default Transition Time Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_GET

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET_UNACK
Chapter 2. API 参考

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_STATUS
Generic Power OnOff Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_GET

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_STATUS
Generic Power OnOff Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET_UNACK
Generic Power Level Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_STATUS
Generic Power Level Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET_UNACK
Generic Battery Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_GET

ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_STATUS
Generic Location Message Opcode
Chapter 2. API

- ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_GET
- ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_STATUS
- ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_GET
- ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_STATUS
  Generic Location Setup Message Opcode
- ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET
- ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET_UNACK
- ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET
- ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET_UNACK
  Generic Manufacturer Property Message Opcode
- ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_GET
- ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_STATUS
- ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_GET
- ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET
- ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET_UNACK
- ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_STATUS
  Generic Admin Property Message Opcode
- ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_GET
- ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_STATUS
- ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_GET
- ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET
- ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET_UNACK
- ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_STATUS
  Generic User Property Message Opcode
- ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_GET
Chapter 2. API 参考

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_STATUS
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_GET
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET_UNACK
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_STATUS
    Generic Client Property Message Opcode
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_GET
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_STATUS
ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET
ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS
ESP_BLE_MESH_MODEL_OP_SENSOR_GET
ESP_BLE_MESH_MODEL_OP_SENSOR_STATUS
ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_GET
ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_STATUS
ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_GET
ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_STATUS
    Sensor Setup Message Opcode
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_STATUS
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_STATUS
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_GET
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET_UNACK
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_STATUS
ESP_BLE_MESH_MODEL_OP_TIME_GET
ESP_BLE_MESH_MODEL_OP_TIME_SET
ESP_BLE_MESH_MODEL_OP_TIME_STATUS
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_GET
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_SET
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_STATUS
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_GET
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_SET
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_STATUS
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_GET
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_SET
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_STATUS
  Scene Message Opcode
ESP_BLE_MESH_MODEL_OP_SCENE_GET
ESP_BLE_MESH_MODEL_OP_SCENE_RECALL
ESP_BLE_MESH_MODEL_OP_SCENE_RECALL_UNACK
ESP_BLE_MESH_MODEL_OP_SCENE_STATUS
ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_GET
ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_STATUS
  Scene Setup Message Opcode
ESP_BLE_MESH_MODEL_OP_SCENE_STORE
ESP_BLE_MESH_MODEL_OP_SCENE_STORE_UNACK

ESP_BLE_MESH_MODEL_OP_SCENE_DELETE

ESP_BLE_MESH_MODEL_OP_SCENE_DELETE_UNACK

Scheduler Message Opcode

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_GET

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_STATUS

ESP_BLE_MESH_MODEL_OP_SCHEDULER_GET

ESP_BLE_MESH_MODEL_OP_SCHEDULER_STATUS

Scheduler Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_STATUS
Light Lightness Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET_UNACK
Light CTL Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_DEFAULT_STATUS
Light CTL Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET_UNACK
Light HSL Message Opcode
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_GET
Light HSL Setup Message Opcode
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK
Light xyL Message Opcode
Chapter 2. API

- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_GET
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET_UNACK
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_STATUS
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_GET
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_STATUS
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_GET
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_STATUS
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_GET
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_STATUS
  - Light xyL Setup Message Opcode
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET_UNACK
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET
- ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET_UNACK
  - Light Control Message Opcode
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_GET
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET_UNACK
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_STATUS
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_GET
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK
- ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_STATUS
**Chapter 2. API**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_GET**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_STATUS**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_GET**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET_UNACK**

**ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_STATUS**

**ESP_BLE_MESH_MODEL_STATUS_SUCCESS**

**ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MIN**

**ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MAX**

**ESP_BLE_MESH_SERVER_RSP_BY_APP**

Response need to be sent in the application

**ESP_BLE_MESH_SERVER_AUTO_RSP**

Response will be sent internally

**Type Definitions**

typedef uint8_t esp_ble_mesh_octet16_t[ESP_BLE_MESH_OCTET16_LEN]

Define the BLE Mesh octet 8 bytes size

typedef uint8_t esp_ble_mesh_octet8_t[ESP_BLE_MESH_OCTET8_LEN]

Invalid Company ID

typedef uint32_t esp_ble_mesh_cb_t

typedef uint8_t UINT8

typedef uint16_t UINT16

typedef uint32_t UINT32

typedef uint64_t UINT64
typedef \texttt{UINT8} BT\_OCTET32[BT\_OCTET32\_LEN]

typedef uint8\_t BD\_ADDR[BD\_ADDR\_LEN]

typedef uint8\_t esp\_ble\_mesh\_bd\_addr\_t[BD\_ADDR\_LEN]

typedef uint8\_t esp\_ble\_mesh\_addr\_type\_t

BLE device address type.

typedef struct esp\_ble\_mesh\_model esp\_ble\_mesh\_model\_t

typedef uint8\_t esp\_ble\_mesh\_dev\_add\_flag\_t

typedef uint32\_t esp\_ble\_mesh\_opcode\_config\_client\_get\_t

esp\_ble\_mesh\_opcode\_config\_client\_get\_t belongs to esp\_ble\_mesh\_opcode\_t, this typedef is only used to locate the opcodes used by esp\_ble\_mesh\_config\_client\_get\_state. The following opcodes will only be used in the esp\_ble\_mesh\_config\_client\_get\_state function.

typedef uint32\_t esp\_ble\_mesh\_opcode\_config\_client\_set\_t

esp\_ble\_mesh\_opcode\_config\_client\_set\_t belongs to esp\_ble\_mesh\_opcode\_t, this typedef is only used to locate the opcodes used by esp\_ble\_mesh\_config\_client\_set\_state. The following opcodes will only be used in the esp\_ble\_mesh\_config\_client\_set\_state function.

typedef uint32\_t esp\_ble\_mesh\_opcode\_config\_status\_t

esp\_ble\_mesh\_opcode\_config\_status\_t belongs to esp\_ble\_mesh\_opcode\_t, this typedef is only used to locate the opcodes used by esp\_ble\_mesh\_config\_client\_get\_state. The following opcodes are used by the BLE Mesh Config Server Model internally to respond to the Config Client Model’s request messages.

typedef uint8\_t esp\_ble\_mesh\_cfg\_status\_t

This typedef is only used to indicate the status code contained in some of the Configuration Server Model status message.

typedef uint32\_t esp\_ble\_mesh\_opcode\_health\_client\_get\_t

esp\_ble\_mesh\_opcode\_health\_client\_get\_t belongs to esp\_ble\_mesh\_opcode\_t, this typedef is only used to locate the opcodes used by esp\_ble\_mesh\_health\_client\_get\_state. The following opcodes will only be used in the esp\_ble\_mesh\_health\_client\_get\_state function.

typedef uint32\_t esp\_ble\_mesh\_opcode\_health\_client\_set\_t

esp\_ble\_mesh\_opcode\_health\_client\_set\_t belongs to esp\_ble\_mesh\_opcode\_t, this typedef is only used to locate the opcodes used by esp\_ble\_mesh\_health\_client\_set\_state. The following opcodes will only be used in the esp\_ble\_mesh\_health\_client\_set\_state function.

typedef uint32\_t esp\_ble\_mesh\_health\_model\_status\_t

esp\_ble\_mesh\_health\_model\_status\_t belongs to esp\_ble\_mesh\_opcode\_t, this typedef is only used to locate the opcodes used by the Health Model messages. The following opcodes are used by the BLE Mesh Health Server Model internally to respond to the Health Client Model’s request messages.

typedef uint32\_t esp\_ble\_mesh\_generic\_message\_opcode\_t

esp\_ble\_mesh\_generic\_message\_opcode\_t belongs to esp\_ble\_mesh\_opcode\_t, this typedef is only used to locate the opcodes used by functions esp\_ble\_mesh\_generic\_client\_get\_state & esp\_ble\_mesh\_generic\_client\_set\_state. Generic OnOff Message Opcode
typedef uint32_t esp_ble_mesh_sensor_message_opcode_t

esp_ble_mesh_sensor_message_opcode_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by functions esp_ble_mesh_sensor_client_get_state & esp_ble_mesh_sensor_client_set_state. Sensor Message Opcode

typedef uint32_t esp_ble_mesh_time_scene_message_opcode_t

esp_ble_mesh_time_scene_message_opcode_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by functions esp_ble_mesh_time_scene_client_get_state & esp_ble_mesh_time_scene_client_set_state. Time Message Opcode

typedef uint32_t esp_ble_mesh_light_message_opcode_t

esp_ble_mesh_light_message_opcode_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by functions esp_ble_mesh_light_client_get_state & esp_ble_mesh_light_client_set_state. Light Lightness Message Opcode

typedef uint32_t esp_ble_mesh_opcode_t

End of defines of esp_ble_mesh_opcode_t

typedef uint8_t esp_ble_mesh_model_status_t

This typedef is only used to indicate the status code contained in some of the server models (e.g. Generic Server Model) status message.

Enumerations

enum esp_ble_mesh_cb_type_t

Values:

enumerator ESP_BLE_MESH_TYPE_PROV_CB
enumerator ESP_BLE_MESH_TYPE_OUTPUT_NUM_CB
enumerator ESP_BLE_MESH_TYPE_OUTPUT_STR_CB
enumerator ESP_BLE_MESH_TYPE_INPUT_CB
enumerator ESP_BLE_MESH_TYPE_LINK_OPEN_CB
enumerator ESP_BLE_MESH_TYPE_LINK_CLOSE_CB
enumerator ESP_BLE_MESH_TYPE_COMPLETE_CB
enumerator ESP_BLE_MESH_TYPE_RESET_CB

enum esp_ble_mesh_oob_method_t

Values:

enumerator ESP_BLE_MESH_NO_OOB
enumerator ESP_BLE_MESH_STATIC_OOB
enumerator ESP_BLE_MESH_OUTPUT_OOB

enumerator ESP_BLE_MESH_INPUT_OOB

enum esp_ble_mesh_output_action_t
  Values:
  enumerator ESP_BLE_MESH_NO_OUTPUT
  enumerator ESP_BLE_MESH_BLINK
  enumerator ESP_BLE_MESH_BEEP
  enumerator ESP_BLE_MESH_VIBRATE
  enumerator ESP_BLE_MESH_DISPLAY_NUMBER
  enumerator ESP_BLE_MESH_DISPLAY_STRING

enum esp_ble_mesh_input_action_t
  Values:
  enumerator ESP_BLE_MESH_NO_INPUT
  enumerator ESP_BLE_MESH_PUSH
  enumerator ESP_BLE_MESH_TWIST
  enumerator ESP_BLE_MESH_ENTER_NUMBER
  enumerator ESP_BLE_MESH_ENTER_STRING

enum esp_ble_mesh_prov_bearer_t
  Values:
  enumerator ESP_BLE_MESH_PROV_ADV
  enumerator ESP_BLE_MESH_PROV_GATT

enum esp_ble_mesh_prov_oob_info_t
  Values:
  enumerator ESP_BLE_MESH_PROV_OOB_OTHER
  enumerator ESP_BLE_MESH_PROV_OOB_URI
enum ESP_BLE_MESH_PROV_OOB_2D_CODE
enumerator ESP_BLE_MESH_PROV_OOB_BAR_CODE
enumerator ESP_BLE_MESH_PROV_OOB_NFC
enumerator ESP_BLE_MESH_PROV_OOB_NUMBER
enumerator ESP_BLE_MESH_PROV_OOB_STRING
enumerator ESP_BLE_MESH_PROV_OOB_ON_BOX
enumerator ESP_BLE_MESH_PROV_OOB_IN_BOX
enumerator ESP_BLE_MESH_PROV_OOB_ON_PAPER
enumerator ESP_BLE_MESH_PROV_OOB_IN_MANUAL
enumerator ESP_BLE_MESH_PROV_OOB_ON_DEV

enum esp_ble_mesh_dev_role_t
    Values:
        enumerator ROLE_NODE
        enumerator ROLE_PROVISIONER
        enumerator ROLE_FAST_PROV

denum esp_ble_mesh_fast_prov_action_t
    Values:
        enumerator FAST_PROV_ACT_NONE
        enumerator FAST_PROV_ACT_ENTER
        enumerator FAST_PROV_ACT_SUSPEND
        enumerator FAST_PROV_ACT_EXIT
        enumerator FAST_PROV_ACT_MAX

denum esp_ble_mesh_proxy_filter_type_t
    Values:
        enumerator PROXY_FILTER_WHITELIST
Chapter 2. API 参考

enumerator PROXY_FILTER_BLACKLIST

enum esp_ble_mesh_prov_cb_event_t

Values:

enumerator ESP_BLE_MESH_PROV_REGISTER_COMP_EVT
Initialize BLE Mesh provisioning capabilities and internal data information completion event

enumerator ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT
Set the unprovisioned device name completion event

enumerator ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT
Enable node provisioning functionality completion event

enumerator ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT
Disable node provisioning functionality completion event

enumerator ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT
Establish a BLE Mesh link event

enumerator ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT
Close a BLE Mesh link event

enumerator ESP_BLE_MESH_NODE_PROV_OOB_PUB_KEY_EVT
Generate Node input OOB public key event

enumerator ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT
Generate Node Output Number event

enumerator ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT
Generate Node Output String event

enumerator ESP_BLE_MESH_NODE_PROV_INPUT_EVT
Event requiring the user to input a number or string

enumerator ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT
Provisioning done event

enumerator ESP_BLE_MESH_NODE_PROV_RESET_EVT
Provisioning reset event

enumerator ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT
Node set oob public key completion event

enumerator ESP_BLE_MESH_NODE_PROV_INPUT_NUMBER_COMP_EVT
Node input number completion event

enumerator ESP_BLE_MESH_NODE_PROV_INPUT_STRING_COMP_EVT
Node input string completion event
enumerator **ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT**
Enable BLE Mesh Proxy Identity advertising completion event

denumerator **ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT**
Enable BLE Mesh GATT Proxy Service completion event

denumerator **ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT**
Disable BLE Mesh GATT Proxy Service completion event

denumerator **ESP_BLE_MESH_NODE_ADD_LOCAL_NET_KEY_COMP_EVT**
Node add NetKey locally completion event

denumerator **ESP_BLE_MESH_NODE_ADD_LOCAL_APP_KEY_COMP_EVT**
Node add AppKey locally completion event

denumerator **ESP_BLE_MESH_NODE_BIND_APP_KEY_TO_MODEL_COMP_EVT**
Node bind AppKey to model locally completion event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT**
Provisioner enable provisioning functionality completion event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT**
Provisioner disable provisioning functionality completion event

denumerator **ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT.EVT**
Provisioner receives unprovisioned device beacon event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_EVT**
Provisioner read unprovisioned device OOB public key event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT**
Provisioner input value for provisioning procedure event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT**
Provisioner output value for provisioning procedure event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT**
Provisioner establish a BLE Mesh link event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT**
Provisioner close a BLE Mesh link event

denumerator **ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT**
Provisioner provisioning done event

denumerator **ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT**
Provisioner add a device to the list which contains devices that are waiting/going to be provisioned completion event
enumerator **ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP_EVT**
Provisioner start to provision an unprovisioned device completion event

enumerator **ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT**
Provisioner delete a device from the list, close provisioning link with the device completion event

enumerator **ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT**
Provisioner set the value to be compared with part of the unprovisioned device UUID completion event

enumerator **ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT**
Provisioner set net_idx/flags/iv_index used for provisioning completion event

enumerator **ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT**
Provisioner set static oob value used for provisioning completion event

enumerator **ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT**
Provisioner set unicast address of primary element completion event

enumerator **ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_COMP_EVT**
Provisioner read unprovisioned device OOB public key completion event

enumerator **ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_EVT**
Provisioner input number completion event

enumerator **ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EVT**
Provisioner input string completion event

enumerator **ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT**
Provisioner set node name completion event

enumerator **ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT**
Provisioner add local app key completion event

enumerator **ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT**
Provisioner update local app key completion event

enumerator **ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT**
Provisioner bind local model with local app key completion event

enumerator **ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_EVT**
Provisioner add local network key completion event

enumerator **ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT**
Provisioner update local network key completion event

enumerator **ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT**
Provisioner store node composition data completion event
Chapter 2. API

enumerator `ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_COMP_EVT`
Provisioner delete node with uuid completion event

denumerator `ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT`
Provisioner delete node with unicast address completion event

denumerator `ESP_BLE_MESH_PROVISIONER_ENABLE_HEARTBEAT_RECV_COMP_EVT`
Provisioner start to receive heartbeat message completion event

enumerator `ESP_BLE_MESH_PROVISIONER_SET_HEARTBEAT_FILTER_TYPE_COMP_EVT`
Provisioner set the heartbeat filter type completion event

enumerator `ESP_BLE_MESH_PROVISIONER_SET_HEARTBEAT_FILTER_INFO_COMP_EVT`
Provisioner set the heartbeat filter information completion event

enumerator `ESP_BLE_MESH_PROVISIONER_RECV_HEARTBEAT_MESSAGE_EVT`
Provisioner receive heartbeat message event

enumerator `ESP_BLE_MESH_PROVISIONER_DIRECT_ERASE_SETTINGS_COMP_EVT`
Provisioner directly erase settings completion event

enumerator `ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_INDEX_COMP_EVT`
Provisioner open settings with index completion event

enumerator `ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_UID_COMP_EVT`
Provisioner open settings with userid completion event

enumerator `ESP_BLE_MESH_PROVISIONER_CLOSE_SETTINGS_WITH_INDEX_COMP_EVT`
Provisioner close settings with index completion event

enumerator `ESP_BLE_MESH_PROVISIONER_CLOSE_SETTINGS_WITH_UID_COMP_EVT`
Provisioner close settings with userid completion event

enumerator `ESP_BLE_MESH_PROVISIONER_DELETE_SETTINGS_WITH_INDEX_COMP_EVT`
Provisioner delete settings with index completion event

enumerator `ESP_BLE_MESH_PROVISIONER_DELETE_SETTINGS_WITH_UID_COMP_EVT`
Provisioner delete settings with userid completion event

enumerator `ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT`
Set fast provisioning information (e.g. unicast address range, net_idx, etc.) completion event

enumerator `ESP_BLE_MESH_SET_FAST_PROV_ACTION_COMP_EVT`
Set fast provisioning action completion event

enumerator `ESP_BLE_MESH_HEARTBEAT_MESSAGE_RECV_EVT`
Receive Heartbeat message event
enumerator ESP_BLE_MESH_LPACIÓN_ENABLE_COMP_EVT
   Enable Low Power Node completion event

enumerator ESP_BLE_MESH_LPÓN_DISABLE_COMP_EVT
   Disable Low Power Node completion event

enumerator ESP_BLE_MESH_LPÓN_POLL_COMP_EVT
   Low Power Node send Friend Poll completion event

enumerator ESP_BLE_MESH_LPÓN_FRIENDSHIP_ESTABLISH_EVT
   Low Power Node establishes friendship event

enumerator ESP_BLE_MESH_LPÓN_FRIENDSHIP_TERMINATE_EVT
   Low Power Node terminates friendship event

enumerator ESP_BLE_MESH_FRIEND_FRIENDSHIP_ESTABLISH_EVT
   Friend Node establishes friendship event

enumerator ESP_BLE_MESH_FRIEND_FRIENDSHIP_TERMINATE_EVT
   Friend Node terminates friendship event

enumerator ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT
   Proxy Client receives Network ID advertising packet event

enumerator ESP_BLE_MESH_PROXY_CLIENT_CONNECTED_EVT
   Proxy Client establishes connection successfully event

enumerator ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT
   Proxy Client terminates connection successfully event

enumerator ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT
   Proxy Client receives Proxy Filter Status event

enumerator ESP_BLE_MESH_PROXY_CLIENT_CONNECT_COMP_EVT
   Proxy Client connect completion event

enumerator ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT
   Proxy Client disconnect completion event

enumerator ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EVT
   Proxy Client set filter type completion event

enumerator ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EVT
   Proxy Client add filter address completion event

enumerator ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT
   Proxy Client remove filter address completion event
enumerator ESP_BLE_MESH_PROXY_SERVER_CONNECTED_EVT
Proxy Server establishes connection successfully event

enumerator ESP_BLE_MESH_PROXY_SERVER_DISCONNECTED_EVT
Proxy Server terminates connection successfully event

enumerator ESP_BLE_MESH_MODEL_SUBSCRIBE_GROUP_ADDR_COMP_EVT
Local model subscribes group address completion event

enumerator ESP_BLE_MESH_MODEL_UNSUBSCRIBE_GROUP_ADDR_COMP_EVT
Local model unsubscribes group address completion event

enumerator ESP_BLE_MESH_DEINIT_MESH_COMP_EVT
De-initialize BLE Mesh stack completion event

enumerator ESP_BLE_MESH_PROV_EVT_MAX

enum [anonymous]
BLE Mesh server models related definitions.
This enum value is the flag of transition timer operation
Values:

enumerator ESP_BLE_MESH_SERVER_TRANS_TIMER_START

enumerator ESP_BLE_MESH_SERVER_FLAG_MAX

enum esp_ble_mesh_server_state_type_t
This enum value is the type of server model states
Values:

enumerator ESP_BLE_MESH_GENERIC_ONOFF_STATE

enumerator ESP_BLE_MESH_GENERIC_LEVEL_STATE

enumerator ESP_BLE_MESH_GENERIC_ONPOWERUP_STATE

enumerator ESP_BLE_MESH_GENERIC_POWER_ACTUAL_STATE

enumerator ESP_BLE_MESH_LIGHT_LIGHTESTHNESS_ACTUAL_STATE

enumerator ESP_BLE_MESH_LIGHT_LIGHTESTHNESS_LINEAR_STATE

enumerator ESP_BLE_MESH_LIGHT_CTL_LIGHTESTHNESS_STATE

enumerator ESP_BLE_MESH_LIGHT_CTL_TEMP_DELTA_UV_STATE
enumerator ESP_BLE_MESH_LIGHT_HSL_STATE

denumerator ESP_BLE_MESH_LIGHT_HSL_LIGHTNESS_STATE

denumerator ESP_BLE_MESH_LIGHT_HSL_HUE_STATE

denumerator ESP_BLE_MESH_LIGHT_HSL_SATURATION_STATE

denumerator ESP_BLE_MESH_LIGHT_XYL_LIGHTNESS_STATE

denumerator ESP_BLE_MESH_LIGHT_LC_LIGHT_ONOFF_STATE

denumerator ESP_BLE_MESH_SERVER_MODEL_STATE_MAX

enum esp_ble_mesh_model_cb_event_t

Values:

denumerator ESP_BLE_MESH_MODEL_OPERATION_EVT
User-defined models receive messages from peer devices (e.g. get, set, status, etc) event

denumerator ESP_BLE_MESH_MODEL_SEND_COMP_EVT
User-defined models send messages completion event

denumerator ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT
User-defined models publish messages completion event

denumerator ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT
User-defined client models receive publish messages event

denumerator ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT
Timeout event for the user-defined client models that failed to receive response from peer server models

denumerator ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT
When a model is configured to publish messages periodically, this event will occur during every publish period

denumerator ESP_BLE_MESH_SERVER_MODEL_UPDATE_STATE_COMP_EVT
Server models update state value completion event

denumerator ESP_BLE_MESH_MODEL_EVT_MAX

ESP-BLE-MESH Core API Reference

This section contains ESP-BLE-MESH Core related APIs, which can be used to initialize ESP-BLE-MESH stack, provision, send/publish messages, etc.

This API reference covers six components:

• ESP-BLE-MESH Stack Initialization
Chapter 2. API 参考

- Reading of Local Data Information
- Low Power Operation (Updating)
- Send/Publish Messages, add Local AppKey, etc.
- ESP-BLE-MESH Node/Provisioner Provisioning
- ESP-BLE-MESH GATT Proxy Server

ESP-BLE-MESH Stack Initialization

Header File
- components/bt/esp_ble_mesh/api/core/include/esp_ble_mesh_common_api.h

Functions

```c
esp_err_t esp_ble_mesh_init(esp_ble_mesh_prov_t *prov, esp_ble_mesh_comp_t *comp)
```

Initialize BLE Mesh module. This API initializes provisioning capabilities and composition data information.

- **prov** - [in] Pointer to the device provisioning capabilities. This pointer must remain valid during the lifetime of the BLE Mesh device.
- **comp** - [in] Pointer to the device composition data information. This pointer must remain valid during the lifetime of the BLE Mesh device.

备注: After calling this API, the device needs to call esp_ble_mesh_prov_enable() to enable provisioning functionality again.

```c
esp_err_t esp_ble_mesh_deinit(esp_ble_mesh_deinit_param_t *param)
```

De-initialize BLE Mesh module.

备注: This function shall be invoked after esp_ble_mesh_client_model_deinit().

- **param** - [in] Pointer to the structure of BLE Mesh deinit parameters.

Reading of Local Data Information

Header File
- components/bt/esp_ble_mesh/api/core/include/esp_ble_mesh_local_data_operation_api.h

Functions

```c
int32_t esp_ble_mesh_get_model_publish_period(esp_ble_mesh_model_t *model)
```

Get the model publish period, the unit is ms.


```c
uint16_t esp_ble_mesh_get_primary_element_address(void)
```

Get the address of the primary element.

备注: Address of the primary element on success, or ESP_BLE_MESH_ADDR_UNASSIGNED on failure which means the device has not been provisioned.
uint16_t *esp_ble_mesh_is_model_subscribed_to_group (esp_ble_mesh_model_t *model, uint16_t group_addr)

检查模型是否已订阅到给定的组地址。注意：例如，一旦收到状态消息，且目标地址是一个组地址，模型便使用此 API 来检查它是否成功订阅到给定的组地址。

参数
- model [in] 模型指针。
- group_addr [in] 组地址。

返回 指向模型的订阅列表中成功订阅组地址的指针，或者在失败时指向 NULL，这意味着模型尚未订阅到给定的组地址。注意：通过返回的组地址指针，您可以将组地址重置为 0x0000 以取消订阅模型。

esp_ble_mesh_elem_t *esp_ble_mesh_find_element (uint16_t element_addr)

通过元素地址找到 BLE 网络元素指针。

参数 element_addr [in] 元素地址。

返回 成功时指向元素的指针，否则返回 NULL。

uint8_t esp_ble_mesh_get_element_count (void)

获取已注册的元素数量。

返回 元素数量。

esp_ble_mesh_model_t *esp_ble_mesh_find_vendor_model (const esp_ble_mesh_elem_t *element, uint16_t company_id, uint16_t model_id)

根据给定的元素、公司 ID 和 Vendor Model ID 查找 Vendor 特定模型。

参数
- element [in] 属于模型的元素。
- company_id [in] 由 Bluetooth SIG 分配的 16 位公司标识符。
- model_id [in] 由厂商分配的 16 位模型标识符。

返回 成功时指向 Vendor Model 的指针，否则返回 NULL，指示 Vendor Model 未找到。

esp_ble_mesh_model_t *esp_ble_mesh_find_sig_model (const esp_ble_mesh_elem_t *element, uint16_t model_id)

根据给定的元素和 Model ID 查找 SIG 模型。

参数
- element [in] 属于模型的元素。
- model_id [in] SIG 模型标识符。

返回 成功时指向 SIG Model 的指针，否则返回 NULL，指示 SIG Model 未找到。

const esp_ble_mesh_comp_t *esp_ble_mesh_get_composition_data (void)

获取已注册的 Composition 数据。

返回 成功时指向 Composition 数据的指针，否则返回 NULL，指示 Composition 数据未初始化。

esp_err_t esp_ble_mesh_model_subscribe_group_addr (uint16_t element_addr, uint16_t company_id, uint16_t model_id, uint16_t group_addr)

本地节点或 Provisioner 订阅组地址。

参数
- element_addr [in] 属于模型的元素的单播地址。

备注：此函数在节点未被配置或 Provisioner 启动之前不应被调用。
• `company_id` [in] A 16-bit company identifier.
• `group_addr` [in] The group address to be subscribed.

Annotation: This function shall not be invoked before node is provisioned or Provisioner is enabled.

```c
esp_err_t esp_ble_mesh_model_unsubscribe_group_addr (uint16_t element_addr, uint16_t company_id, uint16_t model_id, uint16_t group_addr)
```

A local model of node or Provisioner unsubscribes a group address.

```
const uint8_t* esp_ble_mesh_node_get_local_net_key (uint16_t net_idx)
```

This function is called by Node to get the local NetKey.

```
const uint8_t* esp_ble_mesh_node_get_local_app_key (uint16_t app_idx)
```

This function is called by Node to get the local AppKey.

```
esp_err_t esp_ble_mesh_node_add_local_net_key (const uint8_t net_key[16], uint16_t net_idx)
```

This function is called by Node to add a local NetKey.

Annotation: This function can only be called after the device is provisioned.

```
esp_err_t esp_ble_mesh_node_add_local_app_key (const uint8_t app_key[16], uint16_t net_idx, uint16_t app_idx)
```

This function is called by Node to add a local AppKey.

Annotation: The net_idx must be an existing one. This function can only be called after the device is provisioned.
**esp_err_t** esp_ble_mesh_node_bind_app_key_to_local_model (uint16_t element_addr, uint16_t company_id, uint16_t model_id, uint16_t app_idx)

This function is called by Node to bind AppKey to model locally.

**备注:** If going to bind app_key with local vendor model, the company_id shall be set to 0xFFFF. This function can only be called after the device is provisioned.

**参数**
- **element_addr** - [in] Node local element address
- **company_id** - [in] Node local company id
- **model_id** - [in] Node local model id
- **app_idx** - [in] Node local appkey index

返回 ESP_OK on success or error code otherwise.

### Low Power Operation (Updating)

**Header File**
- components/bt/esp_ble_mesh/api/core/include/esp_ble_mesh_low_power_api.h

**Functions**

```c
esp_err_t esp_ble_mesh_lpn_enable (void)
```

Enable BLE Mesh device LPN functionality.

**备注:** This API enables LPN functionality. Once called, the proper Friend Request will be sent.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_lpn_disable (bool force)
```

Disable BLE Mesh device LPN functionality.

**参数**
- **force** - [in] when disabling LPN functionality, use this flag to indicate whether directly clear corresponding information or just send friend clear to disable it if friendship has already been established.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_lpn_poll (void)
```

LPN tries to poll messages from the Friend Node.

**备注:** The Friend Poll message is sent by a Low Power node to ask the Friend node to send a message that it has stored for the Low Power node. Users can call this API to send Friend Poll message manually. If this API is not invoked, the bottom layer of the Low Power node will send Friend Poll before the PollTimeout timer expires. If the corresponding Friend Update is received and MD is set to 0, which means there are no messages for the Low Power node, then the Low Power node will stop scanning.

返回 ESP_OK on success or error code otherwise.

**Send/Publish Messages, add Local AppKey, etc.**
Header File

- components/bt/esp_ble_mesh/api/core/include/esp_ble_mesh_networking_api.h

Functions

`esp_err_t esp_ble_mesh_register_custom_model_callback (esp_ble_mesh_model_cb_t callback)`

Register BLE Mesh callback for user-defined models’ operations. This callback can report the following events generated for the user-defined models:

- Call back the messages received by user-defined client and server models to the application layer;
- If users call esp_ble_mesh_server/client_model_send, this callback notifies the application layer of the send_complete event;
- If user-defined client model sends a message that requires response, and the response message is received after the timer expires, the response message will be reported to the application layer as published by a peer device;
- If the user-defined client model fails to receive the response message during a specified period of time, a timeout event will be reported to the application layer.

**备注:** The client models (i.e. Config Client model, Health Client model, Generic Client models, Sensor Client model, Scene Client model and Lighting Client models) that have been realized internally have their specific register functions. For example, esp_ble_mesh_register_config_client_callback is the register function for Config Client Model.

**参数**
- callback - [in] Pointer to the callback function.
- 返回 ESP_OK on success or error code otherwise.

`esp_err_t esp_ble_mesh_model_msg_opcode_init (uint8_t *data, uint32_t opcode)`

Add the message opcode to the beginning of the model message before sending or publishing the model message.

**备注:** This API is only used to set the opcode of the message.

**参数**
- data - [in] Pointer to the message data.
- 返回 ESP_OK on success or error code otherwise.

`esp_err_t esp_ble_mesh_client_model_init (esp_ble_mesh_model_t *model)`

Initialize the user-defined client model. All user-defined client models shall call this function to initialize the client model internal data. Node: Before calling this API, the op_pair_size and op_pair variabled within the user_data(defined using esp_ble_mesh_client_t_) of the client model need to be initialized.

**参数**
- model - [in] BLE Mesh Client model to which the message belongs.
- 返回 ESP_OK on success or error code otherwise.

`esp_err_t esp_ble_mesh_client_model_deinit (esp_ble_mesh_model_t *model)`

De-initialize the user-defined client model.

**备注:** This function shall be invoked before esp_ble_mesh_deinit() is called.

**参数**
- model - [in] Pointer of the Client model.
- 返回 ESP_OK on success or error code otherwise.
### Chapter 2. API 参考

**esp_err_t esp_ble_mesh_server_model_send_msg**

```c
esp_err_t esp_ble_mesh_server_model_send_msg(esp_ble_mesh_model_t *model,
                                           esp_ble_mesh_msg_ctx_t *ctx, uint32_t opcode, 
                                           uint16_t length, uint8_t *data)
```

Send server model messages (such as server model status messages).

**参数**
- `model` - [in] BLE Mesh Server Model to which the message belongs.
- `ctx` - [in] Message context, includes keys, TTL, etc.
- `length` - [in] Message length (exclude the message opcode).
- `data` - [in] Parameters of Access Payload (exclude the message opcode) to be sent.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_client_model_send_msg**

```c
esp_err_t esp_ble_mesh_client_model_send_msg(esp_ble_mesh_model_t *model,
                                            esp_ble_mesh_msg_ctx_t *ctx, uint32_t opcode, 
                                            uint16_t length, uint8_t *data, int32_t msg_timeout, 
                                            bool need_rsp, esp_ble_mesh_dev_role_t device_role)
```

Send client model messages (such as model get, set, etc).

**参数**
- `model` - [in] BLE Mesh Client Model to which the message belongs.
- `ctx` - [in] Message context, includes keys, TTL, etc.
- `length` - [in] Message length (exclude the message opcode).
- `data` - [in] Parameters of the Access Payload (exclude the message opcode) to be sent.
- `msg_timeout` - [in] Time to get response to the message (in milliseconds).
- `need_rsp` - [in] TRUE if the opcode requires the peer device to reply, FALSE otherwise.
- `device_role` - [in] Role of the device (Node/Provisioner) that sends the message.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_model_publish**

```c
esp_err_t esp_ble_mesh_model_publish(esp_ble_mesh_model_t *model, uint32_t opcode, uint16_t length, uint8_t *data, esp_ble_mesh_dev_role_t device_role)
```

Send a model publication message.

**备注：** Before calling this function, the user needs to ensure that the model publication message (`esp_ble_mesh_model_pub_t::msg`) contains a valid message to be sent. And if users want to update the publishing message, this API should be called in ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT with the message updated.

**参数**
- `length` - [in] Message length (exclude the message opcode).
- `data` - [in] Parameters of the Access Payload (exclude the message opcode) to be sent.
- `device_role` - [in] Role of the device (node/provisioner) publishing the message of the type `esp_ble_mesh_dev_role_t`.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_server_model_update_state**

```c
esp_err_t esp_ble_mesh_server_model_update_state(esp_ble_mesh_model_t *model, 
                                                esp_ble_mesh_server_state_type_t type, 
                                                esp_ble_mesh_server_state_value_t *value)
```

Update a server model state value. If the model publication state is set properly (e.g. publish address is set to a valid address), it will publish corresponding status message.
Chapter 2. API 参考

### esp_err_t esp_ble_mesh_node_local_reset (void)
Reset the provisioning procedure of the local BLE Mesh node.

**备注:** All provisioning information in this node will be deleted and the node needs to be reprovisioned. The API function esp_ble_mesh_node_prov_enable() needs to be called to start a new provisioning procedure.

**参数**
- **index** - [in] Index of the node in the node queue.
- **name** - [in] Name (end by ‘\0’) to be set for the node.

**返回** ESP_OK on success or error code otherwise.

### esp_ble_mesh_provisioner_set_node_name (uint16_t index, const char *name)
This function is called to set the node (provisioned device) name.

**备注:** index is obtained from the parameters of ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT.

**参数**
- **index** - [in] Index of the node in the node queue.
- **name** - [in] Name (end by ‘\0’) to be set for the node.

**返回** ESP_OK on success or error code otherwise.

### esp_ble_mesh_provisioner_get_node_name (uint16_t index)
This function is called to get the node (provisioned device) name.

**备注:** index is obtained from the parameters of ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT.

**参数**
- **index** - [in] Index of the node in the node queue.

**返回** Node name on success, or NULL on failure.

### esp_ble_mesh_provisioner_get_node_index (const char *name)
This function is called to get the node (provisioned device) index.

**参数**
- **name** - [in] Name of the node (end by ‘\0’).

**返回** Node index on success, or an invalid value (0xFFFF) on failure.

### esp_ble_mesh_provisioner_store_node_comp_data (uint16_t unicast_addr, uint8_t *data, uint16_t length)
This function is called to store the Composition Data of the node.

**参数**
- **unicast_addr** - [in] Element address of the node
- **data** - [in] Pointer of Composition Data
- **length** - [in] Length of Composition Data

**返回** ESP_OK on success or error code otherwise.
esp_ble_mesh_node_t *esp_ble_mesh_provisioner_get_node_with_uuid (const uint8_t uuid[16])

This function is called to get the provisioned node information with the node device uuid.

参数 uuid - [in] Device UUID of the node
返回 Pointer of the node info struct or NULL on failure.

esp_ble_mesh_node_t *esp_ble_mesh_provisioner_get_node_with_addr (uint16_t unicast_addr)

This function is called to get the provisioned node information with the node unicast address.

参数 unicast_addr - [in] Unicast address of the node
返回 Pointer of the node info struct or NULL on failure.

esp_ble_mesh_node_t *esp_ble_mesh_provisioner_get_node_with_name (const char *name)

This function is called to get the provisioned node information with the node name.

参数 name - [in] Name of the node (end by \0).
返回 Pointer of the node info struct or NULL on failure.

uint16_t esp_ble_mesh_provisioner_get_prov_node_count (void)

This function is called by Provisioner to get provisioned node count.

返回 Number of the provisioned nodes.

const esp_ble_mesh_node_t **esp_ble_mesh_provisioner_get_node_table_entry (void)

This function is called by Provisioner to get the entry of the node table.

备注: After invoking the function to get the entry of nodes, users can use the “for” loop combined with the macro CONFIG_BLE_MESH_MAX_PROV_NODES to get each node’s information. Before trying to read the node’s information, users need to check if the node exists, i.e. if the *(esp_ble_mesh_node_t **node) is NULL. For example: “

const esp_ble_mesh_node_t **entry = esp_ble_mesh_provisioner_get_node_table_entry(); for (int i = 0; i < CONFIG_BLE_MESH_MAX_PROV_NODES; i++) { const esp_ble_mesh_node_t *node = entry[i]; if (node) { …… } ”

返回 Pointer to the start of the node table.

esp_err_t esp_ble_mesh_provisioner_delete_node_with_uuid (const uint8_t uuid[16])

This function is called to delete the provisioned node information with the node device uuid.

参数 uuid - [in] Device UUID of the node
返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_delete_node_with_addr (uint16_t unicast_addr)

This function is called to delete the provisioned node information with the node unicast address.

参数 unicast_addr - [in] Unicast address of the node
返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_add_local_app_key (const uint8_t app_key[16], uint16_t net_idx, uint16_t app_idx)

This function is called to add a local AppKey for Provisioner.

备注: app_key: If set to NULL, app_key will be generated internally. net_idx: Should be an existing one. app_idx: If it is going to be generated internally, it should be set to 0xFFFF, and the new app_idx will be reported via an event.

参数
- app_key - [in] The app key to be set for the local BLE Mesh stack.
### Chapter 2. API 參考

- **app_idx** - [in] The app key index.
  
  返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_provisioner_update_local_app_key**

(const uint8_t app_key[16],
uint16_t net_idx, uint16_t app_idx)

This function is used to update a local AppKey for Provisioner.

参数

- **app_key** - [in] Value of the AppKey.
- **net_idx** - [in] Corresponding NetKey Index.
- **app_idx** - [in] The AppKey Index

返回 ESP_OK on success or error code otherwise.

**esp_ble_mesh_provisioner_get_local_app_key**

(const uint8_t* esp_ble_mesh_provisioner_get_local_app_key (uint16_t net_idx, uint16_t app_idx)

This function is called by Provisioner to get the local app key value.

参数

- **net_idx** - [in] Network key index.
- **app_idx** - [in] Application key index.

返回 App key on success, or NULL on failure.

**esp_err_t esp_ble_mesh_provisioner_bind_app_key_to_local_model**

(const uint16_t element_addr,
uint16_t app_idx,
uint16_t model_id,
uint16_t company_id)

This function is called by Provisioner to bind own model with proper app key.

备注：company_id: If going to bind app_key with local vendor model, company_id should be set to 0xFFFF.

参数

- **element_addr** - [in] Provisioner local element address
- **app_idx** - [in] Provisioner local appkey index
- **model_id** - [in] Provisioner local model id
- **company_id** - [in] Provisioner local company id

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_provisioner_add_local_net_key**

(const uint8_t net_key[16], uint16_t net_idx)

This function is called by Provisioner to add local network key.

备注: net_key: If set to NULL, net_key will be generated internally. net_idx: If it is going to be generated internally, it should be set to 0xFFFF, and the new net_idx will be reported via an event.

参数

- **net_key** - [in] The network key to be added to the Provisioner local BLE Mesh stack.
- **net_idx** - [in] The network key index.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_provisioner_update_local_net_key**

(const uint8_t net_key[16],
uint16_t net_idx)

This function is called by Provisioner to update a local network key.

参数

- **net_key** - [in] Value of the NetKey.
Chapter 2. API 参考

- **net_idx** - [in] The NetKey Index.
  返回 ESP_OK on success or error code otherwise.

```c
const uint8_t* esp_ble_mesh_provisioner_get_local_net_key (uint16_t net_idx)
```

This function is called by Provisioner to get the local network key value.

- 返回 Network key on success, or NULL on failure.

```c
esp_err_t esp_ble_mesh_provisioner_recv_heartbeat (bool enable)
```

This function is called by Provisioner to enable or disable receiving heartbeat messages.

备注： If enabling receiving heartbeat message successfully, the filter will be an empty rejectlist by default, which means all heartbeat messages received by the Provisioner will be reported to the application layer.

- 参数 **enable** - [in] Enable or disable receiving heartbeat messages.
- 返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_set_heartbeat_filter_type (uint8_t type)
```

This function is called by Provisioner to set the heartbeat filter type.

备注： 1. If the filter type is not the same with the current value, then all the filter entries will be cleaned.
   a. If the previous type is rejectlist, and changed to acceptlist, then the filter will be an empty acceptlist, which means no heartbeat messages will be reported. Users need to add SRC or DST into the filter entry, then heartbeat messages from the SRC or to the DST will be reported.

- 参数 **type** - [in] Heartbeat filter type (acceptlist or rejectlist).
- 返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_set_heartbeat_filter_info (uint8_t op, esp_ble_mesh_heartbeat_filter_info_t* info)
```

This function is called by Provisioner to add or remove a heartbeat filter entry.

a. If the operation is “REMOVE”, the “hb_src” can be set to the SRC (can only be a unicast address) of heartbeat messages, and the “hb_dst” can be set to the DST (unicast address or group address), at least one of them needs to be set.
   • The filter entry with the same SRC or DST will be removed.

备注： 1. If the operation is “ADD”, the “hb_src” can be set to the SRC (can only be a unicast address) of heartbeat messages, and the “hb_dst” can be set to the DST (unicast address or group address), at least one of them needs to be set.
   • If only one of them is set, the filter entry will only use the configured SRC or DST to filter heartbeat messages.
   • If both of them are set, the SRC and DST will both be used to decide if a heartbeat message will be handled.
   • If SRC or DST already exists in some filter entry, then the corresponding entry will be cleaned firstly, then a new entry will be allocated to store the information.

- 参数
  - **op** - [in] Add or REMOVE
Chapter 2. API

• info - [in] Heartbeat filter entry information, including: hb_src - Heartbeat source address; hb_dst - Heartbeat destination address;

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_direct_erase_settings (void)

This function is called by Provisioner to directly erase the mesh information from nvs namespace.

备注: This function can be invoked when the mesh stack is not initialized or has been de-initialized.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_open_settings_with_index (uint8_t index)

This function is called by Provisioner to open a nvs namespace for storing mesh information.

备注: Before open another nvs namespace, the previously opened nvs namespace must be closed firstly.

参数 index - [in] Settings index.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_open_settings_with_uid (const char *uid)

This function is called by Provisioner to open a nvs namespace for storing mesh information.

备注: Before open another nvs namespace, the previously opened nvs namespace must be closed firstly.

参数 uid - [in] Settings userid.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_close_settings_with_index (uint8_t index, bool erase)

This function is called by Provisioner to close a nvs namespace which is opened previously for storing mesh information.

备注: 1. Before closing the nvs namespace, it must be open.

a. When the function is invoked, the Provisioner functionality will be disabled firstly, and: a) If the “erase” flag is set to false, the mesh information will be cleaned (e.g. removing NetKey, AppKey, nodes, etc) from the mesh stack. b) If the “erase” flag is set to true, the mesh information stored in the nvs namespace will also be erased besides been cleaned from the mesh stack.

b. If Provisioner tries to work properly again, we can invoke the open function to open a new nvs namespace or a previously added one, and restore the mesh information from it if not erased.

c. The working process shall be as following: a) Open settings A b) Start to provision and control nodes c) Close settings A d) Open settings B e) Start to provision and control other nodes f) Close settings B g) ... ...

参数

• index - [in] Settings index.

• erase - [in] Indicate if erasing mesh information.

返回 ESP_OK on success or error code otherwise.
esp_err_t esp_ble_mesh_provisioner_close_settings_with_uid(const char *uid, bool erase)

This function is called by Provisioner to close a nvs namespace which is opened previously for storing mesh information.

备注: 1. Before closing the nvs namespace, it must be open.
   a. When the function is invoked, the Provisioner functionality will be disabled firstly, and: a) If the “erase” flag is set to false, the mesh information will be cleaned (e.g. removing NetKey, AppKey, nodes, etc) from the mesh stack. b) If the “erase” flag is set to true, the mesh information stored in the nvs namespace will also be erased besides been cleaned from the mesh stack.
   b. If Provisioner tries to work properly again, we can invoke the open function to open a new nvs namespace or a previously added one, and restore the mesh information from it if not erased.
   c. The working process shall be as following: a) Open settings A b) Start to provision and control nodes c) Close settings A d) Open settings B e) Start to provision and control other nodes f) Close settings B g) ...

参数
   * **uid** – [in] Settings user id.
   * **erase** – [in] Indicate if erasing mesh information.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_delete_settings_with_index(uint8_t index)

This function is called by Provisioner to erase the mesh information and settings user id from a nvs namespace.

备注: When this function is called, the nvs namespace must not be open. This function is used to erase the mesh information and settings user id which are not used currently.

参数 **index** – [in] Settings index.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_provisioner_delete_settings_with_uid(const char *uid)

This function is called by Provisioner to erase the mesh information and settings user id from a nvs namespace.

备注: When this function is called, the nvs namespace must not be open. This function is used to erase the mesh information and settings user id which are not used currently.

参数 **uid** – [in] Settings user id.

返回 ESP_OK on success or error code otherwise.

const char *esp_ble_mesh_provisioner_get_settings_uid(uint8_t index)

This function is called by Provisioner to get settings user id.

参数 **index** – [in] Settings index.

返回 Setting user id on success or NULL on failure.

uint8_t esp_ble_mesh_provisioner_get_settings_index(const char *uid)

This function is called by Provisioner to get settings index.

参数 **uid** – [in] Settings user id.

返回 Settings index.

uint8_t esp_ble_mesh_provisioner_get_free_settings_count(void)

This function is called by Provisioner to get the number of free settings user id.

返回 Number of free settings user id.
const uint8_t *esp_ble_mesh_get_fast_prov_app_key (uint16_t net_idx, uint16_t app_idx)
This function is called to get fast provisioning application key.

参数

返回 Application key on success, or NULL on failure.

Type Definitions
typedef void (*esp_ble_mesh_model_cb_t)(esp_ble_mesh_model_cb_event_t event,
    esp_ble_mesh_model_cb_param_t *param)
: event, event code of user-defined model events; param, parameters of user-defined model events

ESP-BLE-MESH Node/Provisioner Provisioning

Header File
- components/bt/esp_ble_mesh/api/core/include/esp_ble_mesh_provisioning_api.h

Functions
esp_err_t esp_ble_mesh_register_prov_callback (esp_ble_mesh_prov_cb_t callback)
Register BLE Mesh provisioning callback.

参数 callback [in] Pointer to the callback function.

返回 ESP_OK on success or error code otherwise.

bool esp_ble_mesh_node_is_provisioned (void)
Check if a device has been provisioned.

返回 TRUE if the device is provisioned, FALSE if the device is unprovisioned.

esp_err_t esp_ble_mesh_node_prov_enable (esp_ble_mesh_prov_bearer_t bearers)
Enable specific provisioning bearers to get the device ready for provisioning.


esp_err_t esp_ble_mesh_node_prov_disable (esp_ble_mesh_prov_bearer_t bearers)
Disable specific provisioning bearers to make a device inaccessible for provisioning.

参数 bearers  Bit-wise OR of provisioning bearers.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_node_set_oob_pub_key (uint8_t pub_key_x[32], uint8_t pub_key_y[32],
    uint8_t private_key[32])
Unprovisioned device set own oob public key & private key pair.

备注: In order to avoid suffering brute-forcing attack (CVE-2020-26559). The Bluetooth SIG recommends that potentially vulnerable mesh provisioners use an out-of-band mechanism to exchange the public keys. So as an unprovisioned device, it should use this function to input the Public Key exchanged through the out-of-band mechanism.
**Chapter 2. API Reference**

**esp_err_t esp_ble_mesh_node_input_number** (uint32_t number)

Provide provisioning input OOB number.

**参数** number - [in] Number input by device.

**备注:** This is intended to be called if the user has received ESP_BLE_MESH_NODE_PROV_INPUT_EVT with ESP_BLE_MESH_ENTER_NUMBER as the action.

**esp_err_t esp_ble_mesh_node_input_string** (const char* string)

Provide provisioning input OOB string.

**参数** string - [in] String input by device.

**备注:** This is intended to be called if the user has received ESP_BLE_MESH_NODE_PROV_INPUT_EVT with ESP_BLE_MESH_ENTER_STRING as the action.

**esp_err_t esp_ble_mesh_set_unprovisioned_device_name** (const char* name)

Using this function, an unprovisioned device can set its own device name, which will be broadcasted in its advertising data.

**参数** name - [in] Unprovisioned device name

**备注:** This API applicable to PB-GATT mode only by setting the name to the scan response data, it doesn’t apply to PB-ADV mode.

**esp_err_t esp_ble_mesh_provisioner_read_oob_pub_key** (uint8_t link_idx, uint8_t pub_key_x[32],
uint8_t pub_key_y[32])

Provisioner inputs unprovisioned device’s OOB public key.

**备注:** In order to avoid suffering brute-forcing attack (CVE-2020-26559). The Bluetooth SIG recommends that potentially vulnerable mesh provisioners use an out-of-band mechanism to exchange the public keys.

**参数**
- link_idx - [in] The provisioning link index
- pub_key_x - [in] Unprovisioned device’s Public Key X
- pub_key_y - [in] Unprovisioned device’s Public Key Y

**备注:**
This is intended to be called after the esp_ble_mesh_prov_t prov_input_num callback has been called with ESP_BLE_MESH_ENTER_STRING as the action.

**Parameters**

- **string** [in] String input by Provisioner.
- **link_idx** [in] The provisioning link index.

**Return** ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_input_number (uint32_t number, uint8_t link_idx)
```

Provide provisioning input OOB number.

This is intended to be called after the esp_ble_mesh_prov_t prov_input_num callback has been called with ESP_BLE_MESH_ENTER_NUMBER as the action.

**Parameters**

- **number** [in] Number input by Provisioner.
- **link_idx** [in] The provisioning link index.

**Return** ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_prov_enable (esp_ble_mesh_prov_bearer_t bearers)
```

Enable one or more provisioning bearers.

**Note:** PB-ADV: Enable BLE scan. PB-GATT: Initialize corresponding BLE Mesh Proxy info.

**Parameters**

- **bearers** [in] Bit-wise OR of provisioning bearers.

**Return** ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_prov_disable (esp_ble_mesh_prov_bearer_t bearers)
```

Disable one or more provisioning bearers.

**Note:** PB-ADV: Disable BLE scan. PB-GATT: Break any existing BLE Mesh Provisioning connections.

**Parameters**

- **bearers** [in] Bit-wise OR of provisioning bearers.

**Return** ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_add_unprov_dev (esp_ble_mesh_unprov_dev_add_t *add_dev, esp_ble_mesh_dev_add_flag_t flags)
```

Add unprovisioned device info to the unprov_dev queue.

**Note:** 1. Currently address type only supports public address and static random address.

a. If device UUID and/or device address as well as address type already exist in the device queue, but the bearer is different from the existing one, add operation will also be successful and it will update the provision bearer supported by the device.

b. For example, if the Provisioner wants to add an unprovisioned device info before receiving its unprovisioned device beacon or Mesh Provisioning advertising packets, the Provisioner can use this API to add
the device info with each one or both of device UUID and device address added. When the Provisioner gets the device’s advertising packets, it will start provisioning the device internally.

- In this situation, the Provisioner can set bearers with each one or both of ESP_BLE_MESH_PROV_ADV and ESP_BLE_MESH_PROV_GATT enabled, and cannot set flags with ADD_DEV_START_PROV_NOW_FLAG enabled.

- Another example is when the Provisioner receives the unprovisioned device’s beacon or Mesh Provisioning advertising packets, the advertising packets will be reported on to the application layer using the callback registered by the function esp_ble_mesh_register_prov_callback. And in the callback, the Provisioner can call this API to start provisioning the device.

- If the Provisioner uses PB-ADV to provision, either one or both of device UUID and device address can be added, bearers shall be set with ESP_BLE_MESH_PROV_ADV enabled and the flags shall be set with ADD_DEV_START_PROV_NOW_FLAG enabled.

- If the Provisioner uses PB-GATT to provision, both the device UUID and device address need to be added, bearers shall be set with ESP_BLE_MESH_PROV_GATT enabled, and the flags shall be set with ADD_DEV_START_PROV_NOW_FLAG enabled.

- If the Provisioner just wants to store the unprovisioned device info when receiving its advertising packets and start to provision it the next time (e.g. after receiving its advertising packets again), then it can add the device info with either one or both of device UUID and device address included. Bearers can be set with either one or both of ESP_BLE_MESH_PROV_ADV and ESP_BLE_MESH_PROV_GATT enabled (recommend to enable the bearer which will receive its advertising packets, because if the other bearer is enabled, the Provisioner is not aware if the device supports the bearer), and flags cannot be set with ADD_DEV_START_PROV_NOW_FLAG enabled.

- Note: ESP_BLE_MESH_PROV_ADV, ESP_BLE_MESH_PROV_GATT and ADD_DEV_START_PROV_NOW_FLAG cannot be enabled at the same time.

**参数**

- add_dev – [in] Pointer to a struct containing the device information
- flags – [in] Flags indicate several operations on the device information
  - Remove device information from queue after device has been provisioned (BIT0)
  - Start provisioning immediately after device is added to queue (BIT1)
  - Device can be removed if device queue is full (BIT2)

**返回**

ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_prov_device_with_addr(
    const uint8_t uuid[16],
    esp_ble_mesh_bd_addr_t addr,
    esp_ble_mesh_addr_type_t addr_type,
    esp_ble_mesh_prov_bearer_t bearer,
    uint16_t oob_info,
    uint16_t unicast_addr)
```

Provision an unprovisioned device and assign a fixed unicast address for it in advance.

**备注**

1. Currently address type only supports public address and static random address.

a. Bearer must be equal to ESP_BLE_MESH_PROV_ADV or ESP_BLE_MESH_PROV_GATT, since Provisioner will start to provision a device immediately once this function is invoked. And the input bearer must be identical with the one within the parameters of the ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT event.

b. If this function is used by a Provisioner to provision devices, the application should take care of the assigned unicast address and avoid overlap of the unicast addresses of different nodes.

c. Recommend to use only one of the functions “esp_ble_mesh_provisioner_add_unprov_dev” and “esp_ble_mesh_provisioner_prov_device_with_addr” by a Provisioner.

**参数**

- uuid – [in] Device UUID of the unprovisioned device
Chapter 2. API

- **addr** - [in] Device address of the unprovisioned device
- **addr_type** - [in] Device address type of the unprovisioned device
- **bearer** - [in] Provisioning bearer going to be used by Provisioner
- **oob_info** - [in] OOB info of the unprovisioned device
- **unicast_addr** - [in] Unicast address going to be allocated for the unprovisioned device

备注: Zero on success or (negative) error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_delete_dev(esp_ble_mesh_device_delete_t *del_dev)
```

Delete device from queue, and reset current provisioning link with the device.

参数 **del_dev** - [in] Pointer to a struct containing the device information.

备注: If the device is in the queue, remove it from the queue; if the device is being provisioned, terminate the provisioning procedure. Either one of the device address or device UUID can be used as input.

```c
esp_err_t esp_ble_mesh_provisioner_set_dev_uuid_match(const uint8_t*match_val,uint8_t match_len,uint8_t offset,bool prov_after_match)
```

This function is called by Provisioner to set the part of the device UUID to be compared before starting to provision.

参数
- **match_val** - [in] Value to be compared with the part of the device UUID.
- **match_len** - [in] Length of the compared match value.
- **offset** - [in] Offset of the device UUID to be compared (based on zero).
- **prov_after_match** - [in] Flag used to indicate whether provisioner should start to provision the device immediately if the part of the UUID matches.

备注: ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_set_prov_data_info(esp_ble_mesh_prov_data_info_t *prov_data_info)
```

This function is called by Provisioner to set provisioning data information before starting to provision.

参数 **prov_data_info** - [in] Pointer to a struct containing net_idx or flags or iv_index.

备注: ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_provisioner_set_static_oob_value(const uint8_t*value,uint8_t length)
```

This function is called by Provisioner to set static oob value used for provisioning.

AuthValues selected using a cryptographically secure random or pseudorandom number generator and having the maximum permitted entropy (128-bits) will be most difficult to brute-force. AuthValues with reduced entropy or generated in a predictable manner will not grant the same level of protection against this vulnerability. Selecting a new AuthValue with each provisioning attempt can also make it more difficult to launch a brute-force attack by requiring the attacker to restart the search with each provisioning attempt (CVE-2020-26556).

备注: The Bluetooth SIG recommends that mesh implementations enforce a randomly selected AuthValue using all of the available bits, where permitted by the implementation. A large entropy helps ensure that a brute-force of the AuthValue, even a static AuthValue, cannot normally be completed in a reasonable time (CVE-2020-26557).

备注: The Bluetooth SIG recommends that mesh implementations enforce a randomly selected AuthValue using all of the available bits, where permitted by the implementation. A large entropy helps ensure that a brute-force of the AuthValue, even a static AuthValue, cannot normally be completed in a reasonable time (CVE-2020-26557).

参数
- **value** - [in] Pointer to the static oob value.
- **length** - [in] Length of the static oob value.
esp_err_t esp_ble_mesh_provisioner_set_primary_elem_addr(uint16_t addr)

This function is called by Provisioner to set own Primary element address.

- **Parameters**
  - `addr` [in] Unicast address of the Primary element of Provisioner.
  - Return ESP_OK on success or error code otherwise.

**Note:** This API must be invoked when BLE Mesh initialization is completed successfully, and can be invoked before Provisioner functionality is enabled. Once this API is invoked successfully, the prov_unicast_addr value in the struct `esp_ble_mesh_prov_t` will be ignored, and Provisioner will use this address as its own primary element address. And if the unicast address going to assigned for the next unprovisioned device is smaller than the input address + element number of Provisioner, then the address for the next unprovisioned device will be recalculated internally.

**Type Definitions**

typedef void (*esp_ble_mesh_prov_cb_t)(esp_ble_mesh_prov_cb_event_t event,
                              esp_ble_mesh_prov_cb_param_t *param)

: event, event code of provisioning events; param, parameters of provisioning events

typedef void (*esp_ble_mesh_adv_cb_t)(const esp_ble_mesh_bd_addr_t addr,
                                    const esp_ble_mesh_addr_type_t addr_type,
                                    const uint8_t adv_type,
                                    const uint8_t *dev_uuid,
                                    esp_ble_mesh_prov_bearer_t bearer)

Callback for Provisioner that received advertising packets from unprovisioned devices which are not in the unprovisioned device queue.

Report on the unprovisioned device beacon and mesh provisioning service adv data to application.

- **Parameters**
  - `addr` [in] Pointer to the unprovisioned device address.
  - `addr_type` [in] Unprovisioned device address type.
  - `adv_type` [in] Adv packet type(ADV_IND or ADV_NONCONN_IND).
  - `dev_uuid` [in] Unprovisioned device UUID pointer.
  - `oob_info` [in] OOB information of the unprovisioned device.
  - `bearer` [in] Adv packet received from PB-GATT or PB-ADV bearer.

**ESP-BLE-MESH GATT Proxy Server**

**Header File**

- components/bt/esp_ble_mesh/api/core/include/esp_ble_mesh_proxy_api.h
Functions

```c
esp_err_t esp_ble_mesh_proxy_identity_enable(void)
```

Enable advertising with Node Identity.

备注：此API要求GATT Proxy支持被启用。一旦调用，每个子网开始使用Node Identity广告60秒，60秒后网络ID将被广告。在正常情况下，BLE Mesh Proxy Node Identity和Network ID广告将自动由BLE Mesh堆栈在设备被配置后启用。

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_proxy_gatt_enable(void)
```

Enable BLE Mesh GATT Proxy Service.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_proxy_gatt_disable(void)
```

Disconnect the BLE Mesh GATT Proxy connection if there is any, and disable the BLE Mesh GATT Proxy Service.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_proxy_client_connect(esp_ble_mesh_bd_addr_t addr, esp_ble_mesh_addr_type_t addr_type, uint16_t net_idx)
```

Proxy Client creates a connection with the Proxy Server.

参数
- `addr` - [in] Device address of the Proxy Server.
- `addr_type` - [in] Device address type (public or static random).
- `net_idx` - [in] NetKey Index related with Network ID in the Mesh Proxy advertising packet.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_proxy_client_disconnect(uint8_t conn_handle)
```

Proxy Client terminates a connection with the Proxy Server.

参数 `conn_handle` - [in] Proxy connection handle.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_proxy_client_set_filter_type(uint8_t conn_handle, uint16_t net_idx, esp_ble_mesh_proxy_filter_type_t filter_type)
```

Proxy Client sets the filter type of the Proxy Server.

参数
- `conn_handle` - [in] Proxy connection handle.
- `net_idx` - [in] Corresponding NetKey Index.
- `filter_type` - [in] whitelist or blacklist.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_proxy_client_add_filter_addr(uint8_t conn_handle, uint16_t net_idx, uint16_t *addr, uint16_t addr_num)
```

Proxy Client adds address to the Proxy Server filter list.

参数
- `conn_handle` - [in] Proxy connection handle.
- `net_idx` - [in] Corresponding NetKey Index.
- `addr` - [in] Pointer to the filter address.
- `addr_num` - [in] Number of the filter address.

返回 ESP_OK on success or error code otherwise.
esp_err_t esp_ble_mesh_proxy_client_remove_filter_addr(uint8_t conn_handle, uint16_t net_idx, uint16_t *addr, uint16_t addr_num)

Proxy Client removes address from the Proxy Server filter list.

参数
  • conn_handle [in] Proxy connection handle.
  • net_idx [in] Corresponding NetKey Index.
  • addr [in] Pointer to the filter address.
  • addr_num [in] Number of the filter address.

返回 ESP_OK on success or error code otherwise.

ESP-BLE-MESH Models API Reference

This section contains ESP-BLE-MESH Model related APIs, event types, event parameters, etc.

There are six categories of models:
  • Configuration Client/Server Models
  • Health Client/Server Models
  • Generic Client/Server Models
  • Sensor Client/Server Models
  • Time and Scenes Client/Server Models
  • Lighting Client/Server Models

备注：Definitions related to Server Models are being updated, and will be released soon.

Configuration Client/Server Models

Header File

  • components/bt/esp_ble_mesh/api/models/include/esp_ble_mesh_config_model_api.h

Functions

esp_err_t esp_ble_mesh_register_config_client_callback(esp_ble_mesh_cfg_client_cb_t callback)

Register BLE Mesh Config Client Model callback.

参数 callback [in] Pointer to the callback function.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_register_config_server_callback(esp_ble_mesh_cfg_server_cb_t callback)

Register BLE Mesh Config Server Model callback.

参数 callback [in] Pointer to the callback function.

返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_config_client_get_state(esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_cfg_client_get_state_t *get_state)

Get the value of Config Server Model states using the Config Client Model get messages.

备注：If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_opcode_config_client_get_t in esp_ble_mesh_defs.h
参数
- `params` [in] Pointer to BLE Mesh common client parameters.
- `set_state` [in] Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

包括 `esp_ble_mesh_cfg_client_get_state_t`

```
union esp_ble_mesh_cfg_client_get_state_t
#include <esp_ble_mesh_config_model_api.h> For
ESP_BLE_MESH_MODEL_OP_BEACON_GET
ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_GET
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_MODEL_PUB_GET
ESP_BLE_MESH_MODEL_OP_FRIEND_GET
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_GET
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_GET
```

Union成员
- `esp_ble_mesh_cfg_model_pub_get_t` model_pub_get
  For ESP_BLE_MESH_MODEL_OP_MODEL_PUB_GET.
- `esp_ble_mesh_cfg_composition_data_get_t` comp_data_get
  For ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET.
- `esp_ble_mesh_cfg_sig_model_sub_get_t` sig_model_sub_get
  For ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_GET
- `esp_ble_mesh_cfg_vnd_model_sub_get_t` vnd_model_sub_get
  For ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_GET
- `esp_ble_mesh_cfg_app_key_get_t` app_key_get
  For ESP_BLE_MESH_MODEL_OP_APP_KEY_GET.
- `esp_ble_mesh_cfg_node_identity_get_t` node_identity_get
  For ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_GET.

备注： If you want to find the opcodes and corresponding meanings accepted by this API, please refer to `esp_ble_mesh_opcode_config_client_set_t` in `esp_ble_mesh_defs.h`
**Chapter 2. API 参考**

```c
union esp_ble_mesh_cfg_client_set_state_t

#include <esp_ble_mesh_config_model_api.h>

Esp BLE mesh model API

- esp_ble_mesh_cfg_sig_model_app_get_t sig_model_app_get
  For ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_GET

- esp_ble_mesh_cfg_vnd_model_app_get_t vnd_model_app_get
  For ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_GET

- esp_ble_mesh_cfg_key_phase_get_t kr_phase_get
  For ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_GET

- esp_ble_mesh_cfg_lpn_polltimeout_get_t lpn_pollto_get
  For ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_GET

union esp_ble_mesh_cfg_client_set_state_t

#include <esp_ble_mesh_config_model_api.h>
For ESP_BLE_MESH_MODEL_OP_BEACON_SET
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_SET ESP_BLE_MESH_MODEL_OP_MODEL_PUB_SET
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_ADD ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_ADD
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_DELETE
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_OVERWRITE ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_OVERWRITE
ESP_BLE_MESH_MODEL_OP_NET_KEY_ADD ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD
ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND ESP_BLE_MESH_MODEL_OP_NODE_RESET
ESP_BLE_MESH_MODEL_OP_FRIEND_SET ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_SET
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_SET

The set_state parameter in the esp_ble_mesh_config_client_set_state function should not be set to NULL.

**Public Members**

- esp_ble_mesh_cfg_beacon_set_t beacon_set
  For ESP_BLE_MESH_MODEL_OP_BEACON_SET

- esp_ble_mesh_cfg_default_ttl_set_t default_ttl_set
  For ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET

- esp_ble_mesh_cfg_friend_set_t friend_set
  For ESP_BLE_MESH_MODEL_OP_FRIEND_SET

- esp_ble_mesh_cfg_gatt_proxy_set_t gatt_proxy_set
  For ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET

- esp_ble_mesh_cfg_relay_set_t relay_set
  For ESP_BLE_MESH_MODEL_OP_RELAY_SET

- esp_ble_mesh_cfg_net_key_add_t net_key_add
  For ESP_BLE_MESH_MODEL_OP_NET_KEY_ADD

- esp_ble_mesh_cfg_app_key_add_t app_key_add
  For ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD

- esp_ble_mesh_cfg_model_app_bind_t model_app_bind
  For ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND

Espressif Systems  592  Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
esp_ble_mesh_cfg_model_pub_set_t model_pub_set
  For ESP_BLE_MESH_MODEL_OP_MODEL_PUB_SET

esp_ble_mesh_cfg_model_sub_add_t model_sub_add
  For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_ADD

esp_ble_mesh_cfg_model_sub_delete_t model_sub_delete
  For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE

esp_ble_mesh_cfg_model_sub_overwrite_t model_sub_overwrite
  For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_OVERWRITE

esp_ble_mesh_cfg_model_sub_va_add_t model_sub_va_add
  For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_ADD

esp_ble_mesh_cfg_model_sub_va_delete_t model_sub_va_delete
  For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_DELETE

esp_ble_mesh_cfg_model_sub_va_overwrite_t model_sub_va_overwrite
  For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_OVERWRITE

esp_ble_mesh_cfg_heartbeat_pub_set_t heartbeat_pub_set
  For ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_SET

esp_ble_mesh_cfg_heartbeat_sub_set_t heartbeat_sub_set
  For ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_SET

esp_ble_mesh_cfg_model_pub_va_set_t model_pub_va_set
  For ESP_BLE_MESH_MODEL_OP_MODEL_PUB_VIRTUAL_ADDR_SET

esp_ble_mesh_cfg_model_sub_delete_all_t model_sub_delete_all
  For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE_ALL

esp_ble_mesh_cfg_net_key_update_t net_key_update
  For ESP_BLE_MESH_MODEL_OP_NET_KEY_UPDATE

esp_ble_mesh_cfg_net_key_delete_t net_key_delete
  For ESP_BLE_MESH_MODEL_OP_NET_KEY_DELETE

esp_ble_mesh_cfg_app_key_update_t app_key_update
  For ESP_BLE_MESH_MODEL_OP_APP_KEY_UPDATE

esp_ble_mesh_cfg_app_key_delete_t app_key_delete
  For ESP_BLE_MESH_MODEL_OP_APP_KEY_DELETE

esp_ble_mesh_cfg_node_identity_set_t node_identity_set
  For ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_SET
esp_ble_mesh_cfg_model_app_unbind_t \textbf{model\_app\_unbind}
For ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_APP\_UNBIND

esp_ble_mesh_cfg_kr_phase_set_t \textbf{kr\_phase\_set}
For ESP\_BLE\_MESH\_MODEL\_OP\_KEY\_REFRESH\_PHASE\_SET

esp_ble_mesh_cfg_net_transmit_set_t \textbf{net\_transmit\_set}
For ESP\_BLE\_MESH\_MODEL\_OP\_NETWORK\_TRANSMIT\_SET

union \textbf{esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t}
#include <esp\_ble\_mesh\_config\_model\_api\_h> Configuration Client Model received message union.

\textbf{Public Members}

\textbf{esp\_ble\_mesh\_cfg\_beacon\_status\_cb\_t \textbf{beacon\_status}}
The beacon status value

\textbf{esp\_ble\_mesh\_cfg\_comp\_data\_status\_cb\_t \textbf{comp\_data\_status}}
The composition data status value

\textbf{esp\_ble\_mesh\_cfg\_default\_ttl\_status\_cb\_t \textbf{default\_ttl\_status}}
The default_ttl status value

\textbf{esp\_ble\_mesh\_cfg\_gatt\_proxy\_status\_cb\_t \textbf{gatt\_proxy\_status}}
The gatt_proxy status value

\textbf{esp\_ble\_mesh\_cfg\_relay\_status\_cb\_t \textbf{relay\_status}}
The relay status value

\textbf{esp\_ble\_mesh\_cfg\_model\_pub\_status\_cb\_t \textbf{model\_pub\_status}}
The model publication status value

\textbf{esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t \textbf{model\_sub\_status}}
The model subscription status value

\textbf{esp\_ble\_mesh\_cfg\_net\_key\_status\_cb\_t \textbf{netkey\_status}}
The netkey status value

\textbf{esp\_ble\_mesh\_cfg\_app\_key\_status\_cb\_t \textbf{appkey\_status}}
The appkey status value

\textbf{esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t \textbf{model\_app\_status}}
The model app status value

\textbf{esp\_ble\_mesh\_cfg\_friend\_status\_cb\_t \textbf{friend\_status}}
The friend status value
The heartbeat publication status value

The heartbeat subscription status value

The network transmit status value

The model subscription list value

The network key index list value

The application key index list value

The node identity status value

The model application key index list value

The key refresh phase status value

The low power node poll timeout status value

Configuration Server model state change value union.

Public Members

The `recv_op` in `ctx` can be used to decide which state is changed. Config Model Publication Set

Config Model Subscription Add

Config Model Subscription Delete

Config NetKey Add
Chapter 2. API


definitions:

- `esp_ble_mesh_state_change_cfg_netkey_update_t netkey_update`
  - Config NetKey Update

- `esp_ble_mesh_state_change_cfg_netkey_delete_t netkey_delete`
  - Config NetKey Delete

- `esp_ble_mesh_state_change_cfg_appkey_add_t appkey_add`
  - Config AppKey Add

- `esp_ble_mesh_state_change_cfg_appkey_update_t appkey_update`
  - Config AppKey Update

- `esp_ble_mesh_state_change_cfg_appkey_delete_t appkey_delete`
  - Config AppKey Delete

- `esp_ble_mesh_state_change_cfg_model_app_bind_t mod_app_bind`
  - Config Model App Bind

- `esp_ble_mesh_state_change_cfg_model_app_unbind_t mod_app_unbind`
  - Config Model App Unbind

- `esp_ble_mesh_state_change_cfg_kr_phase_set_t kr_phase_set`
  - Config Key Refresh Phase Set

union `esp_ble_mesh_cfg_server_cb_value_t`

#include <esp_ble_mesh_config_model_api.h> Configuration Server model callback value union.

Public Members

- `esp_ble_mesh_cfg_server_state_change_t state_change`
  
  ESP_BLE_MESH_CFG_SERVER_STATE_CHANGE_EVT

Structures

struct `esp_ble_mesh_cfg_srv`

Configuration Server Model context

Public Members

- `esp_ble_mesh_model_t *model`
  
  Pointer to Configuration Server Model

uint8_t `net_transmit`

  Network Transmit state

uint8_t `relay`

  Relay Mode state
**uint8_t relay_retransmit**
Relay Retransmit state

**uint8_t beacon**
Secure Network Beacon state

**uint8_t gatt_proxy**
GATT Proxy state

**uint8_t friend_state**
Friend state

**uint8_t default_ttl**
Default TTL

**struct k_delayed_work timer**
Heartbeat Publication timer

**uint16_t dst**
Destination address for Heartbeat messages

**uint16_t count**
Number of Heartbeat messages to be sent
Number of Heartbeat messages received

**uint8_t period**
Period for sending Heartbeat messages

**uint8_t ttl**
TTL to be used when sending Heartbeat messages

**uint16_t feature**
Bit field indicating features that trigger Heartbeat messages when changed

**uint16_t net_idx**
NetKey Index used by Heartbeat Publication

**struct esp_ble_mesh_cfg_srv::[anonymous] heartbeat_pub**
Heartbeat Publication

**int64_t expiry**
Timestamp when Heartbeat subscription period is expired

**uint16_t src**
Source address for Heartbeat messages

**uint8_t min_hops**
Minimum hops when receiving Heartbeat messages
uint8_t \texttt{max\_hops}  
Maximum hops when receiving Heartbeat messages

\textit{esp\_ble\_mesh\_cb} \texttt{t} \texttt{heartbeat\_recv\_cb}  
Optional heartbeat subscription tracking function

\textit{struct} \textit{esp\_ble\_mesh\_cfg\_srv}::[\texttt{anonymous}] \texttt{heartbeat\_sub}  
Heartbeat Subscription

\textit{struct} \textit{esp\_ble\_mesh\_cfg\_composition\_data\_get\_t}  
Parameters of Config Composition Data Get.

\textbf{Public Members}

uint8_t \texttt{page}  
Page number of the Composition Data.

\textit{struct} \textit{esp\_ble\_mesh\_cfg\_model\_pub\_get\_t}  
Parameters of Config Model Publication Get.

\textbf{Public Members}

uint16_t \texttt{element\_addr}  
The element address

uint16_t \texttt{model\_id}  
The model id

uint16_t \texttt{company\_id}  
The company id, if not a vendor model, shall set to 0xFFFF

\textit{struct} \textit{esp\_ble\_mesh\_cfg\_sig\_model\_sub\_get\_t}  
Parameters of Config SIG Model Subscription Get.

\textbf{Public Members}

uint16_t \texttt{element\_addr}  
The element address

uint16_t \texttt{model\_id}  
The model id

\textit{struct} \textit{esp\_ble\_mesh\_cfg\_vnd\_model\_sub\_get\_t}  
Parameters of Config Vendor Model Subscription Get.
Chapter 2. API 参考

Public Members

uint16_t element_addr
The element address

uint16_t model_id
The model id

uint16_t company_id
The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_app_key_get_t
Parameters of Config AppKey Get.

Public Members

uint16_t net_idx
The network key index

struct esp_ble_mesh_cfg_node_identity_get_t
Parameters of Config Node Identity Get.

Public Members

uint16_t net_idx
The network key index

struct esp_ble_mesh_cfg_sig_model_app_get_t
Parameters of Config SIG Model App Get.

Public Members

uint16_t element_addr
The element address

uint16_t model_id
The model id

struct esp_ble_mesh_cfg_vnd_model_app_get_t
Parameters of Config Vendor Model App Get.

Public Members

uint16_t element_addr
The element address
## Chapter 2. API 参考

### Public Members

#### uint16_t model_id

The model id

#### uint16_t company_id

The company id, if not a vendor model, shall set to 0xFFFF

#### struct esp_ble_mesh_cfg_kr_phase_get_t

Parameters of Config Key Refresh Phase Get.

##### Public Members

#### uint16_t net_idx

The network key index

#### struct esp_ble_mesh_cfg_lpn_polltimeout_get_t

Parameters of Config Low Power Node PollTimeout Get.

##### Public Members

#### uint16_t lpn_addr

The unicast address of the Low Power node

#### struct esp_ble_mesh_cfg_beacon_set_t

Parameters of Config Beacon Set.

##### Public Members

#### uint8_t beacon

New Secure Network Beacon state

#### struct esp_ble_mesh_cfg_default_ttl_set_t

Parameters of Config Default TTL Set.

##### Public Members

#### uint8_t ttl

The default TTL state value

#### struct esp_ble_mesh_cfg_friend_set_t

Parameters of Config Friend Set.

##### Public Members

#### uint8_t friend_state

The friend state value
struct `esp_ble_mesh_cfg_gatt_proxy_set_t`  
Parameters of Config GATT Proxy Set.

**Public Members**

- `uint8_t gatt_proxy`  
The GATT Proxy state value

struct `esp_ble_mesh_cfg_relay_set_t`  
Parameters of Config Relay Set.

**Public Members**

- `uint8_t relay`  
The relay value
- `uint8_t relay_retransmit`  
The relay retransmit value

struct `esp_ble_mesh_cfg_net_key_add_t`  
Parameters of Config NetKey Add.

**Public Members**

- `uint16_t net_idx`  
The network key index
- `uint8_t net_key[16]`  
The network key value

struct `esp_ble_mesh_cfg_app_key_add_t`  
Parameters of Config AppKey Add.

**Public Members**

- `uint16_t net_idx`  
The network key index
- `uint16_t app_idx`  
The app key index
- `uint8_t app_key[16]`  
The app key value

struct `esp_ble_mesh_cfg_model_app_bind_t`  
Parameters of Config Model App Bind.
Public Members

`uint16_t element_addr`
The element address

`uint16_t model_app_idx`
Index of the app key to bind with the model

`uint16_t model_id`
The model id

`uint16_t company_id`
The company id, if not a vendor model, shall set to 0xFFFF

```
struct esp_ble_mesh_cfg_model_pub_set_t
Parameters of Config Model Publication Set.
```

Public Members

`uint16_t element_addr`
The element address

`uint16_t publish_addr`
Value of the publish address

`uint16_t publish_app_idx`
Index of the application key

`bool cred_flag`
Value of the Friendship Credential Flag

`uint8_t publish_ttl`
Default TTL value for the publishing messages

`uint8_t publish_period`
Period for periodic status publishing

`uint8_t publish_retransmit`
Number of retransmissions and number of 50-millisecond steps between retransmissions

`uint16_t model_id`
The model id

`uint16_t company_id`
The company id, if not a vendor model, shall set to 0xFFFF

```
struct esp_ble_mesh_cfg_model_sub_add_t
Parameters of Config Model Subscription Add.
```
### Public Members

**uint16_t element_addr**
The element address

**uint16_t sub_addr**
The address to be added to the Subscription List

**uint16_t model_id**
The model id

**uint16_t company_id**
The company id, if not a vendor model, shall set to 0xFFFF

```c
struct esp_ble_mesh_cfg_model_sub_delete_t
Parameters of Config Model Subscription Delete.
```

### Public Members

**uint16_t element_addr**
The element address

**uint16_t sub_addr**
The address to be removed from the Subscription List

**uint16_t model_id**
The model id

**uint16_t company_id**
The company id, if not a vendor model, shall set to 0xFFFF

```c
struct esp_ble_mesh_cfg_model_sub_overwrite_t
Parameters of Config Model Subscription Overwrite.
```

### Public Members

**uint16_t element_addr**
The element address

**uint16_t sub_addr**
The address to be added to the Subscription List

**uint16_t model_id**
The model id

**uint16_t company_id**
The company id, if not a vendor model, shall set to 0xFFFF
struct `esp_ble_mesh_cfg_model_sub_va_add_t`
Parameters of Config Model Subscription Virtual Address Add.

**Public Members**

- `uint16_t element_addr`
  The element address
- `uint8_t label_uuid[16]`
  The Label UUID of the virtual address to be added to the Subscription List
- `uint16_t model_id`
  The model id
- `uint16_t company_id`
  The company id, if not a vendor model, shall set to 0xFFFF

struct `esp_ble_mesh_cfg_model_sub_va_delete_t`
Parameters of Config Model Subscription Virtual Address Delete.

**Public Members**

- `uint16_t element_addr`
  The element address
- `uint8_t label_uuid[16]`
  The Label UUID of the virtual address to be removed from the Subscription List
- `uint16_t model_id`
  The model id
- `uint16_t company_id`
  The company id, if not a vendor model, shall set to 0xFFFF

struct `esp_ble_mesh_cfg_model_sub_va_overwrite_t`
Parameters of Config Model Subscription Virtual Address Overwrite.

**Public Members**

- `uint16_t element_addr`
  The element address
- `uint8_t label_uuid[16]`
  The Label UUID of the virtual address to be added to the Subscription List
uint16_t **model_id**
The model id

uint16_t **company_id**
The company id, if not a vendor model, shall set to 0xFFFF

struct **esp_ble_mesh_cfg_model_pub_va_set_t**
Parameters of Config Model Publication Virtual Address Set.

**Public Members**

uint16_t **element_addr**
The element address

uint8_t **label_uuid[16]**
Value of the Label UUID publish address

uint16_t **publish_app_idx**
Index of the application key

bool **cred_flag**
Value of the Friendship Credential Flag

uint8_t **publish_ttl**
Default TTL value for the publishing messages

uint8_t **publish_period**
Period for periodic status publishing

uint8_t **publish_retransmit**
Number of retransmissions and number of 50-millisecond steps between retransmissions

uint16_t **model_id**
The model id

uint16_t **company_id**
The company id, if not a vendor model, shall set to 0xFFFF

struct **esp_ble_mesh_cfg_model_sub_delete_all_t**
Parameters of Config Model Subscription Delete All.

**Public Members**

uint16_t **element_addr**
The element address
uint16_t model_id
The model id

uint16_t company_id
The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_net_key_update_t
Parameters of Config NetKey Update.

Public Members

uint16_t net_idx
The network key index

uint8_t net_key[16]
The network key value

struct esp_ble_mesh_cfg_net_key_delete_t
Parameters of Config NetKey Delete.

Public Members

uint16_t net_idx
The network key index

struct esp_ble_mesh_cfg_app_key_update_t
Parameters of Config AppKey Update.

Public Members

uint16_t net_idx
The network key index

uint16_t app_idx
The app key index

uint8_t app_key[16]
The app key value

struct esp_ble_mesh_cfg_app_key_delete_t
Parameters of Config AppKey Delete.

Public Members

uint16_t net_idx
The network key index
Chapter 2. API

```c
uint16_t app_idx
    The app key index

struct esp_ble_mesh_cfg_node_identity_set_t
    Parameters of Config Node Identity Set.
    
Public Members

uint16_t net_idx
    The network key index

uint8_t identity
    New Node Identity state

struct esp_ble_mesh_cfg_model_app_unbind_t
    Parameters of Config Model App Unbind.
    
Public Members

uint16_t element_addr
    The element address

uint16_t model_app_idx
    Index of the app key to bind with the model

uint16_t model_id
    The model id

uint16_t company_id
    The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_kr_phase_set_t
    Parameters of Config Key Refresh Phase Set.
    
Public Members

uint16_t net_idx
    The network key index

uint8_t transition
    New Key Refresh Phase Transition

struct esp_ble_mesh_cfg_net_transmit_set_t
    Parameters of Config Network Transmit Set.
```
Public Members

uint8_t `net_transmit`
Network Transmit State

struct `esp_ble_mesh_cfg_heartbeat_pub_set_t`
Parameters of Config Model Heartbeat Publication Set.

Public Members

uint16_t `dst`
Destination address for Heartbeat messages

uint8_t `count`
Number of Heartbeat messages to be sent

uint8_t `period`
Period for sending Heartbeat messages

uint8_t `ttl`
TTL to be used when sending Heartbeat messages

uint16_t `feature`
Bit field indicating features that trigger Heartbeat messages when changed

uint16_t `net_idx`
NetKey Index

struct `esp_ble_mesh_cfg_heartbeat_sub_set_t`
Parameters of Config Model Heartbeat Subscription Set.

Public Members

uint16_t `src`
Source address for Heartbeat messages

uint16_t `dst`
Destination address for Heartbeat messages

uint8_t `period`
Period for receiving Heartbeat messages

struct `esp_ble_mesh_cfg_beacon_status_cb_t`
Parameter of Config Beacon Status
Public Members

uint8_t beacon
Secure Network Beacon state value

struct esp_ble_mesh_cfg_comp_data_status_cb_t
Parameters of Config Composition Data Status

Public Members

uint8_t page
Page number of the Composition Data

struct net_buf_simple *composition_data
Pointer to Composition Data for the identified page

struct esp_ble_mesh_cfg_default_ttl_status_cb_t
Parameter of Config Default TTL Status

Public Members

uint8_t default_ttl
Default TTL state value

struct esp_ble_mesh_cfg_gatt_proxy_status_cb_t
Parameter of Config GATT Proxy Status

Public Members

uint8_t gatt_proxy
GATT Proxy state value

struct esp_ble_mesh_cfg_relay_status_cb_t
Parameters of Config Relay Status

Public Members

uint8_t relay
Relay state value

uint8_t retransmit
Relay retransmit value(number of retransmissions and number of 10-millisecond steps between retransmissions)

struct esp_ble_mesh_cfg_model_pub_status_cb_t
Parameters of Config Model Publication Status
Public Members

`uint8_t status`
Status Code for the request message

`uint16_t element_addr`
Address of the element

`uint16_t publish_addr`
Value of the publish address

`uint16_t app_idx`
Index of the application key

`bool cred_flag`
Value of the Friendship Credential Flag

`uint8_t ttl`
Default TTL value for the outgoing messages

`uint8_t period`
Period for periodic status publishing

`uint8_t transmit`
Number of retransmissions and number of 50-millisecond steps between retransmissions

`uint16_t company_id`
Company ID

`uint16_t model_id`
Model ID

`struct esp_ble_mesh_cfg_model_sub_status_cb_t`
Parameters of Config Model Subscription Status

Public Members

`uint8_t status`
Status Code for the request message

`uint16_t element_addr`
Address of the element

`uint16_t sub_addr`
Value of the address

`uint16_t company_id`
Company ID
Chapter 2. API 参考

```c
uint16_t model_id
    Model ID

struct esp_ble_mesh_cfg_net_key_status_cb_t
    Parameters of Config NetKey Status

    Public Members

    uint8_t status
        Status Code for the request message

    uint16_t net_idx
        Index of the NetKey

struct esp_ble_mesh_cfg_app_key_status_cb_t
    Parameters of Config AppKey Status

    Public Members

    uint8_t status
        Status Code for the request message

    uint16_t net_idx
        Index of the NetKey

    uint16_t app_idx
        Index of the application key

struct esp_ble_mesh_cfg_mod_app_status_cb_t
    Parameters of Config Model App Status

    Public Members

    uint8_t status
        Status Code for the request message

    uint16_t element_addr
        Address of the element

    uint16_t app_idx
        Index of the application key

    uint16_t company_id
        Company ID
```
**uint16_t model_id**

Model ID

**struct esp_ble_mesh_cfg_friend_status_cb_t**

Parameter of Config Friend Status

**Public Members**

**uint8_t friend_state**

Friend state value

**Struct esp_ble_mesh_cfg_hb_pub_status_cb_t**

Parameters of Config Heartbeat Publication Status

**Public Members**

**uint8_t status**

Status Code for the request message

**uint16_t dst**

Destination address for Heartbeat messages

**uint8_t count**

Number of Heartbeat messages remaining to be sent

**uint8_t period**

Period for sending Heartbeat messages

**uint8_t ttl**

TTL to be used when sending Heartbeat messages

**uint16_t features**

Features that trigger Heartbeat messages when changed

**uint16_t net_idx**

Index of the NetKey

**struct esp_ble_mesh_cfg_hb_sub_status_cb_t**

Parameters of Config Heartbeat Subscription Status

**Public Members**

**uint8_t status**

Status Code for the request message
**Chapter 2. API Reference**

**uint16_t src**
Source address for Heartbeat messages

**uint16_t dst**
Destination address for Heartbeat messages

**uint8_t period**
Remaining Period for processing Heartbeat messages

**uint8_t count**
Number of Heartbeat messages received

**uint8_t min_hops**
Minimum hops when receiving Heartbeat messages

**uint8_t max_hops**
Maximum hops when receiving Heartbeat messages

**struct esp_ble_mesh_cfg_net_trans_status_cb_t**
Parameters of Config Network Transmit Status

**Public Members**

**uint8_t net_trans_count**
Number of transmissions for each Network PDU originating from the node

**uint8_t net_trans_step**
Maximum hops when receiving Heartbeat messages

**struct esp_ble_mesh_cfg_model_sub_list_cb_t**
Parameters of Config SIG/Vendor Subscription List

**Public Members**

**uint8_t status**
Status Code for the request message

**uint16_t element_addr**
Address of the element

**uint16_t company_id**
Company ID

**uint16_t model_id**
Model ID
struct net_buf_simple *\texttt{sub_addr}
A block of all addresses from the Subscription List

struct \texttt{esp\_ble\_mesh\_cfg\_net\_key\_list\_cb\_t}
Parameter of Config NetKey List

\textbf{Public Members}

struct net_buf_simple *\texttt{net\_idx}
A list of NetKey Indexes known to the node

struct \texttt{esp\_ble\_mesh\_cfg\_app\_key\_list\_cb\_t}
Parameters of Config AppKey List

\textbf{Public Members}

uint8_t \texttt{status}
Status Code for the request message

uint16_t \texttt{net\_idx}
NetKey Index of the NetKey that the AppKeys are bound to

struct net_buf_simple *\texttt{app\_idx}
A list of AppKey indexes that are bound to the NetKey identified by NetKeyId

struct \texttt{esp\_ble\_mesh\_cfg\_node\_id\_status\_cb\_t}
Parameters of Config Node Identity Status

\textbf{Public Members}

uint8_t \texttt{status}
Status Code for the request message

uint16_t \texttt{net\_idx}
Index of the NetKey

uint8_t \texttt{identity}
Node Identity state

struct \texttt{esp\_ble\_mesh\_cfg\_model\_app\_list\_cb\_t}
Parameters of Config SIG/Vendor Model App List

\textbf{Public Members}

uint8_t \texttt{status}
Status Code for the request message
uint16_t element_addr
    Address of the element

uint16_t company_id
    Company ID

uint16_t model_id
    Model ID

struct net_buf_simple *app_idx
    All AppKey indexes bound to the Model

struct esp_ble_mesh_cfg_kr_phase_status_cb_t
    Parameters of Config Key Refresh Phase Status

    Public Members

    uint8_t status
        Status Code for the request message

    uint16_t net_idx
        Index of the NetKey

    uint8_t phase
        Key Refresh Phase state

struct esp_ble_mesh_cfg_lpn_pollto_status_cb_t
    Parameters of Config Low Power Node PollTimeout Status

    Public Members

    uint16_t lpn_addr
        The unicast address of the Low Power node

    int32_t poll_timeout
        The current value of the PollTimeout timer of the Low Power node

struct esp_ble_mesh_cfg_client_cb_param_t
    Configuration Client Model callback parameters

    Public Members

    int error_code
        Appropriate error code
**esp_ble_mesh_client_common_param_t** *params

The client common parameters

**esp_ble_mesh_cfg_client_common_cb_param_t** status_cb

The config status message callback values

**struct esp_ble_mesh_state_change_cfg_mod_pub_set_t**

Configuration Server model related context.

**Public Members**

uint16_t **element_addr**

Element Address

uint16_t **pub_addr**

Publish Address

uint16_t **app_idx**

AppKey Index

bool **cred_flag**

Friendship Credential Flag

uint8_t **pub_ttl**

Publish TTL

uint8_t **pub_period**

Publish Period

uint8_t **pub_retransmit**

Publish Retransmit

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

**struct esp_ble_mesh_state_change_cfg_model_sub_add_t**

Parameters of Config Model Subscription Add

**Public Members**

uint16_t **element_addr**

Element Address
Chapter 2. API Reference

.uint16_t sub_addr
    Subscription Address

.uint16_t company_id
    Company ID

.uint16_t model_id
    Model ID

struct esp_ble_mesh_state_change_cfg_netkey_add_t
    Parameters of Config NetKey Add

Public Members

.uint16_t net_idx
    NetKey Index

.uint8_t net_key[16]
    NetKey

struct esp_ble_mesh_state_change_cfg_netkey_update_t
    Parameters of Config NetKey Update

Public Members

.uint16_t net_idx
    NetKey Index
Chapter 2. API 参考

```c
uint8_t net_key[16]
NetKey

struct esp_ble_mesh_state_change_cfg_netkey_delete_t
Parameter of Config NetKey Delete

Public Members

uint16_t net_idx
NetKey Index

struct esp_ble_mesh_state_change_cfg_appkey_add_t
Parameters of Config AppKey Add

Public Members

uint16_t net_idx
NetKey Index

uint16_t app_idx
AppKey Index

uint8_t app_key[16]
AppKey

struct esp_ble_mesh_state_change_cfg_appkey_update_t
Parameters of Config AppKey Update

Public Members

uint16_t net_idx
NetKey Index

uint16_t app_idx
AppKey Index

uint8_t app_key[16]
AppKey

struct esp_ble_mesh_state_change_cfg_appkey_delete_t
Parameters of Config AppKey Delete

Public Members

uint16_t net_idx
NetKey Index
```
Chapter 2. API

\[
\text{uint16_t } \text{app_idx} \\
\quad \text{AppKey Index}
\]

struct \texttt{esp\_ble\_mesh\_state\_change\_cfg\_model\_app\_bind\_t}

Parameters of Config Model App Bind

**Public Members**

\[
\text{uint16_t } \text{element_addr} \\
\quad \text{Element Address}
\]

\[
\text{uint16_t } \text{app_idx} \\
\quad \text{AppKey Index}
\]

\[
\text{uint16_t } \text{company_id} \\
\quad \text{Company ID}
\]

\[
\text{uint16_t } \text{model_id} \\
\quad \text{Model ID}
\]

struct \texttt{esp\_ble\_mesh\_state\_change\_cfg\_model\_app\_unbind\_t}

Parameters of Config Model App Unbind

**Public Members**

\[
\text{uint16_t } \text{element_addr} \\
\quad \text{Element Address}
\]

\[
\text{uint16_t } \text{app_idx} \\
\quad \text{AppKey Index}
\]

\[
\text{uint16_t } \text{company_id} \\
\quad \text{Company ID}
\]

\[
\text{uint16_t } \text{model_id} \\
\quad \text{Model ID}
\]

struct \texttt{esp\_ble\_mesh\_state\_change\_cfg\_kr\_phase\_set\_t}

Parameters of Config Key Refresh Phase Set

**Public Members**

\[
\text{uint16_t } \text{net_idx} \\
\quad \text{NetKey Index}
\]
**uint8_t kr_phase**
New Key Refresh Phase Transition

**struct esp_ble_mesh_cfg_server_cb_param_t**
Configuration Server model callback parameters

**Public Members**

```
esp_ble_mesh_model_t *model
```
Pointer to the server model structure

```
esp_ble_mesh_msg_ctx_t ctx
```
Context of the received message

```
esp_ble_mesh_cfg_server_cb_value_t value
```
Value of the received configuration messages

**Macros**

```
ESP_BLE_MESH_MODEL_CFG_SRV(srv_data)
```
Define a new Config Server Model.

- **参数**
  - `svr_data` - Pointer to a unique Config Server Model user_data.
- **返回**
  - New Config Server Model instance.

```
ESP_BLE_MESH_MODEL_CFG_CLI(cli_data)
```
Define a new Config Client Model.

- **参数**
  - `cli_data` - Pointer to a unique struct esp_ble_mesh_client_t.
- **返回**
  - New Config Client Model instance.

**Type Definitions**

```typedef struct esp_ble_mesh_cfg_srv esp_ble_mesh_cfg_srv_t
```
Configuration Server Model context

```typedef void(*esp_ble_mesh_cfg_client_cb_t)(esp_ble_mesh_cfg_client_cb_event_t event,
esp_ble_mesh_cfg_client_cb_param_t *param)
```
Bluetooth Mesh Config Client and Server Model functions.

- **Configuration Client Model callback function type**

```
Param event Event type
Param param Pointer to callback parameter
```
typedef void (*esp_ble_mesh_cfg_server_cb_t)(
    esp_ble_mesh_cfg_server_cb_event_t event,
    esp_ble_mesh_cfg_server_cb_param_t *param)

Configuration Server Model callback function type.

- **Param event** Event type
- **Param param** Pointer to callback parameter

### Enumerations

**enum esp_ble_mesh_cfg_client_cb_event_t**

This enum value is the event of Configuration Client Model

**Values:**

- enumerator **ESP_BLE_MESH_CFG_CLIENT_GET_STATE_EVT**
- enumerator **ESP_BLE_MESH_CFG_CLIENT_SET_STATE_EVT**
- enumerator **ESP_BLE_MESH_CFG_CLIENT_PUBLISH_EVT**
- enumerator **ESP_BLE_MESH_CFG_CLIENT_TIMEOUT_EVT**
- enumerator **ESP_BLE_MESH_CFG_CLIENT_EVT_MAX**

**enum esp_ble_mesh_cfg_server_cb_event_t**

This enum value is the event of Configuration Server model

**Values:**

- enumerator **ESP_BLE_MESH_CFG_SERVER_STATE_CHANGE_EVT**
- enumerator **ESP_BLE_MESH_CFG_SERVER_EVT_MAX**

### Health Client/Server Models

#### Header File

- components/bt/esp_ble_mesh/api/models/include/esp_ble_mesh_health_model_api.h

#### Functions

**esp_err_t esp_ble_mesh_register_health_client_callback (esp_ble_mesh_health_client_cb_t callback)**

Register BLE Mesh Health Model callback, the callback will report Health Client & Server Model events.

- 参数 **callback** [in] Pointer to the callback function.
- 返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_register_health_server_callback (esp_ble_mesh_health_server_cb_t callback)**

Register BLE Mesh Health Server Model callback.

- 参数 **callback** [in] Pointer to the callback function.
- 返回 ESP_OK on success or error code otherwise.
API

**esp_err_t esp_ble_mesh_health_client_get_state (esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_health_client_get_state_t *get_state)**

This function is called to get the Health Server states using the Health Client Model get messages.

**备注:** If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_opcode_health_client_get_t in esp_ble_mesh_defs.h

参数

- **params** - [in] Pointer to BLE Mesh common client parameters.
- **get_state** - [in] Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_health_client_set_state (esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_health_client_set_state_t *set_state)**

This function is called to set the Health Server states using the Health Client Model set messages.

**备注:** If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_opcode_health_client_set_t in esp_ble_mesh_defs.h

参数

- **params** - [in] Pointer to BLE Mesh common client parameters.
- **set_state** - [in] Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_health_server_fault_update (esp_ble_mesh_elem_t *element)**

This function is called by the Health Server Model to update the context of its Health Current status.

参数 **element** - [in] The element to which the Health Server Model belongs.

返回 ESP_OK on success or error code otherwise.

**Unions**

union **esp_ble_mesh_health_client_get_state_t**

```c
#include <esp_ble_mesh_health_model_api.h>

For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_GET
ESP_BLE_MESH_MODEL_OP_ATTENTION_GET
ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_GET

the get_state parameter in the esp_ble_mesh_health_client_get_state function should not be set to NULL.
```

**Public Members**

union **esp_ble_mesh_health_client_set_state_t**

```c
#include <esp_ble_mesh_health_model_api.h>

For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR_UNACK
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK
ESP_BLE_MESH_MODEL_OP_ATTENTION_SET
ESP_BLE_MESH_MODEL_OP_ATTENTION_SET_UNACK

the set_state parameter in the esp_ble_mesh_health_client_set_state function should not be set to NULL.
```
Public Members

`esp_ble_mesh_health_attention_set_t attention_set`
For ESP_BLE_MESH_MODEL_OP_ATTENTION_SET or ESP_BLE_MESH_MODEL_OP_ATTENTION_SET_UNACK.

`esp_ble_mesh_health_period_set_t period_set`
For ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET or ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET_UNACK.

`esp_ble_mesh_health_fault_test_t fault_test`
For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST or ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK.

`esp_ble_mesh_health_fault_clear_t fault_clear`
For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR or ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR_UNACK.

union `esp_ble_mesh_health_client_common_cb_param_t`
#include <esp_ble_mesh_health_model_api.h> Health Client Model received message union.

Public Members

`esp_ble_mesh_health_current_status_cb_t current_status`
The health current status value

`esp_ble_mesh_health_fault_status_cb_t fault_status`
The health fault status value

`esp_ble_mesh_health_period_status_cb_t period_status`
The health period status value

`esp_ble_mesh_health_attention_status_cb_t attention_status`
The health attention status value

union `esp_ble_mesh_health_server_cb_param_t`
#include <esp_ble_mesh_health_model_api.h> Health Server Model callback parameters union.

Public Members

`esp_ble_mesh_health_fault_update_comp_cb_t fault_update_comp`
ESP_BLE_MESH_HEALTH_SERVER_FAULT_UPDATE_COMP_EVT

`esp_ble_mesh_health_fault_clear_cb_t fault_clear`
ESP_BLE_MESH_HEALTH_SERVER_FAULT_CLEAR_EVT

`esp_ble_mesh_health_fault_test_cb_t fault_test`
ESP_BLE_MESH_HEALTH_SERVER_FAULT_TEST_EVT
**Structures**

```c
struct esp_ble_mesh_health_srv_cb_t
ESP_BLE_MESH_HEALTH_SERVER_CALLBACK
```

### Public Members

- `esp_ble_mesh_cb_t fault_clear`
  Clear health registered faults. Initialized by the stack.

- `esp_ble_mesh_cb_t fault_test`
  Run a specific health test. Initialized by the stack.

- `esp_ble_mesh_cb_t attention_on`
  Health attention on callback. Initialized by the stack.

- `esp_ble_mesh_cb_t attention_off`
  Health attention off callback. Initialized by the stack.

```c
struct esp_ble_mesh_health_test_t
ESP_BLE_MESH_HEALTH_SERVER_TEST_CONTEXT
```

### Public Members

- `uint8_t id_count`
  Number of Health self-test ID

- `const uint8_t *test_ids`
  Array of Health self-test IDs

- `uint16_t company_id`
  Company ID used to identify the Health Fault state

- `uint8_t prev_test_id`
  Current test ID of the health fault test

- `uint8_t current_faults[ESP_BLE_MESH_HEALTH_FAULT_ARRAY_SIZE]`
  Array of current faults

- `uint8_t registered_faults[ESP_BLE_MESH_HEALTH_FAULT_ARRAY_SIZE]`
  Array of registered faults
struct esp_ble_mesh_health_srv_t
    ESP BLE Mesh Health Server Model Context

Public Members

esp_ble_mesh_model_t *model
    Pointer to Health Server Model

esp_ble_mesh_health_srv_cb_t health_cb
    Health callback struct

struct k_delayed_work attention_timer
    Attention Timer state

bool attention_timer_start
    Attention Timer start flag

esp_ble_mesh_health_test_t health_test
    Health Server fault test

struct esp_ble_mesh_health_fault_get_t
    Parameter of Health Fault Get

Public Members

uint16_t company_id
    Bluetooth assigned 16-bit Company ID

struct esp_ble_mesh_health_attention_set_t
    Parameter of Health Attention Set

Public Members

uint8_t attention
    Value of the Attention Timer state

struct esp_ble_mesh_health_period_set_t
    Parameter of Health Period Set

Public Members

uint8_t fast_period_divisor
    Divider for the Publish Period

struct esp_ble_mesh_health_fault_test_t
    Parameter of Health Fault Test
Public Members

\texttt{uint16_t company_id}
Bluetooth assigned 16-bit Company ID

\texttt{uint8_t test_id}
ID of a specific test to be performed

\texttt{struct esp_ble_mesh_health_fault_clear_t}
Parameter of Health Fault Clear

Public Members

\texttt{uint16_t company_id}
Bluetooth assigned 16-bit Company ID

\texttt{struct esp_ble_mesh_health_current_status_cb_t}
Parameters of Health Current Status

Public Members

\texttt{uint8_t test_id}
ID of a most recently performed test

\texttt{uint16_t company_id}
Bluetooth assigned 16-bit Company ID

\texttt{struct net_buf_simple *fault_array}
FaultArray field contains a sequence of 1-octet fault values

\texttt{struct esp_ble_mesh_health_fault_status_cb_t}
Parameters of Health Fault Status

Public Members

\texttt{uint8_t test_id}
ID of a most recently performed test

\texttt{uint16_t company_id}
Bluetooth assigned 16-bit Company ID

\texttt{struct net_buf_simple *fault_array}
FaultArray field contains a sequence of 1-octet fault values

\texttt{struct esp_ble_mesh_health_period_status_cb_t}
Parameter of Health Period Status
Public Members

uint8_t `fast_period_divisor`
Divider for the Publish Period

`struct esp_ble_mesh_health_attention_status_cb_t`
Parameter of Health Attention Status

Public Members

uint8_t `attention`
Value of the Attention Timer state

`struct esp_ble_mesh_health_client_cb_param_t`
Health Client Model callback parameters

Public Members

int `error_code`
Appropriate error code

`esp_ble_mesh_client_common_param_t *params`
The client common parameters.

`esp_ble_mesh_health_client_common_cb_param_t status_cb`
The health message status callback values

`struct esp_ble_mesh_health_fault_update_comp_cb_t`
Parameter of publishing Health Current Status completion event

Public Members

int `error_code`
The result of publishing Health Current Status

`esp_ble_mesh_elem_t *element`
Pointer to the element which contains the Health Server Model

`struct esp_ble_mesh_health_fault_clear_cb_t`
Parameters of Health Fault Clear event

Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Health Server Model
Chapter 2. API

uint16_t company_id
   Bluetooth assigned 16-bit Company ID

struct esp_ble_mesh_health_fault_test_cb_t
   Parameters of Health Fault Test event

   Public Members

   esp_ble_mesh_model_t *model
      Pointer to the Health Server Model

   uint8_t test_id
      ID of a specific test to be performed

   uint16_t company_id
      Bluetooth assigned 16-bit Company ID

struct esp_ble_mesh_health_attention_on_cb_t
   Parameter of Health Attention On event

   Public Members

   esp_ble_mesh_model_t *model
      Pointer to the Health Server Model

   uint8_t time
      Duration of attention timer on (in seconds)

struct esp_ble_mesh_health_attention_off_cb_t
   Parameter of Health Attention Off event

   Public Members

   esp_ble_mesh_model_t *model
      Pointer to the Health Server Model

Macros

ESP_BLE_MESH_MODEL_HEALTH_SRV (srv, pub)
   Define a new Health Server Model.

备注: The Health Server Model can only be included by a Primary Element.

参数
   • srv – Pointer to the unique struct esp_ble_mesh_health_srv_t.
   • pub – Pointer to the unique struct esp_ble_mesh_model_pub_t.

返回 New Health Server Model instance.
Chapter 2. API

**ESP_BLE_MESH_MODEL_HEALTH_CLI** (cli_data)

Define a new Health Client Model.

**备注:** This API needs to be called for each element on which the application needs to have a Health Client Model.

参数
- **cli_data** - Pointer to the unique struct `esp_ble_mesh_client_t`.

返回
New Health Client Model instance.

**ESP_BLE_MESH_HEALTH_PUB_DEFINE** (name, _max, _role)

A helper to define a health publication context

参数
- **name** - Name given to the publication context variable.
- **_max** - Maximum number of faults the element can have.
- **_role** - Role of the device which contains the model.

**ESP_BLE_MESH_HEALTH_STANDARD_TEST**

SIG identifier of Health Fault Test. 0x01~0xFF: Vendor Specific Test.

**ESP_BLE_MESH_NO_FAULT**

Fault values of Health Fault Test. 0x33~0x7F: Reserved for Future Use. 0x80~0xFF: Vendor Specific Warning/Error.

**ESP_BLE_MESH_BATTERY_LOW_WARNING**

**ESP_BLE_MESH_BATTERY_LOW_ERROR**

**ESP_BLE_MESH_SUPPLY_VOLTAGE TOO LOW_WARNING**

**ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_LOW_ERROR**

**ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_HIGH_WARNING**

**ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_HIGH_ERROR**

**ESP_BLE_MESH_POWER_SUPPLY_INTERRUPTED_WARNING**

**ESP_BLE_MESH_POWER_SUPPLY_INTERRUPTED_ERROR**

**ESP_BLE_MESH_NO_LOAD_WARNING**

**ESP_BLE_MESH_NO_LOAD_ERROR**

**ESP_BLE_MESH_OVERLOAD_WARNING**

**ESP_BLE_MESH_OVERLOAD_ERROR**
ESP_BLE_MESH_OVERHEAT_WARNING
ESP_BLE_MESH_OVERHEAT_ERROR
ESP_BLE_MESH_CONDENSATION_WARNING
ESP_BLE_MESH_CONDENSATION_ERROR
ESP_BLE_MESH_VIBRATION_WARNING
ESP_BLE_MESH_VIBRATION_ERROR
ESP_BLE_MESH_CONFIGURATION_WARNING
ESP_BLE_MESH_CONFIGURATION_ERROR
ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_WARNING
ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_ERROR
ESP_BLE_MESH_MEMORY_WARNING
ESP_BLE_MESH_MEMORY_ERROR
ESP_BLE_MESH_SELF_TEST_WARNING
ESP_BLE_MESH_SELF_TEST_ERROR
ESP_BLE_MESH_INPUT_TOO_LOW_WARNING
ESP_BLE_MESH_INPUT_TOO_LOW_ERROR
ESP_BLE_MESH_INPUT_TOO_HIGH_WARNING
ESP_BLE_MESH_INPUT_TOO_HIGH_ERROR
ESP_BLE_MESH_INPUT_NO_CHANGE_WARNING
ESP_BLE_MESH_INPUT_NO_CHANGE_ERROR
ESP_BLE_MESH_ACTUATOR_BLOCKED_WARNING
ESP_BLE_MESH_ACTUATOR_BLOCKED_ERROR
ESP_BLE_MESH_HOUSING_OPENED_WARNING
Chapter 2. API

ESP_BLE_MESH_HOUSING_OPENED_ERROR

ESP_BLE_MESH_TAMPER_WARNING

ESP_BLE_MESH_TAMPER_ERROR

ESP_BLE_MESH_DEVICE_MOVED_WARNING

ESP_BLE_MESH_DEVICE_MOVED_ERROR

ESP_BLE_MESH_DEVICE_DROPPED_WARNING

ESP_BLE_MESH_DEVICE_DROPPED_ERROR

ESP_BLE_MESH_OVERFLOW_WARNING

ESP_BLE_MESH_OVERFLOW_ERROR

ESP_BLE_MESH_EMPTY_WARNING

ESP_BLE_MESH_EMPTY_ERROR

ESP_BLE_MESH_INTERNAL_BUS_WARNING

ESP_BLE_MESH_INTERNAL_BUS_ERROR

ESP_BLE_MESH_MECHANISM_JAMMED_WARNING

ESP_BLE_MESH_MECHANISM_JAMMED_ERROR

ESP_BLE_MESH_HEALTH_FAULT_ARRAY_SIZE

**Type Definitions**

typedef void (*esp_ble_mesh_health_client_cb_t)(esp_ble_mesh_health_client_cb_event_t event,
        esp_ble_mesh_health_client_cb_param_t *param)

    Bluetooth Mesh Health Client and Server Model function.
    Health Client Model callback function type
    
    **Param event** Event type
    **Param param** Pointer to callback parameter

typedef void (*esp_ble_mesh_health_server_cb_t)(esp_ble_mesh_health_server_cb_event_t event,
        esp_ble_mesh_health_server_cb_param_t *param)

    Health Server Model callback function type.
    
    **Param event** Event type
    **Param param** Pointer to callback parameter
Enumerations

enum esp_ble_mesh_health_client_cb_event_t
This enum value is the event of Health Client Model
Values:

enumerator ESP_BLE_MESH_HEALTH_CLIENT_GET_STATE_EVT
enumerator ESP_BLE_MESH_HEALTH_CLIENT_SET_STATE_EVT
enumerator ESP_BLE_MESH_HEALTH_CLIENT_PUBLISH_EVT
enumerator ESP_BLE_MESH_HEALTH_CLIENT_TIMEOUT_EVT
enumerator ESP_BLE_MESH_HEALTH_CLIENT_EVT_MAX

enum esp_ble_mesh_health_server_cb_event_t
This enum value is the event of Health Server Model
Values:

enumerator ESP_BLE_MESH_HEALTH_SERVER_FAULT_UPDATE_COMP_EVT
enumerator ESP_BLE_MESH_HEALTH_SERVER_FAULT_CLEAR_EVT
enumerator ESP_BLE_MESH_HEALTH_SERVER_FAULT_TEST_EVT
enumerator ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_ON_EVT
enumerator ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_OFF_EVT
enumerator ESP_BLE_MESH_HEALTH_SERVER_EVT_MAX

Generic Client/Server Models

Header File
• components/bt/esp_ble_mesh/api/models/include/esp_ble_mesh_generic_model_api.h

Functions

esp_err_t esp_ble_mesh_register_generic_client_callback (esp_ble_mesh_generic_client_cb_t callback)
Register BLE Mesh Generic Client Model callback.
参数 callback - [in] Pointer to the callback function.
返回 ESP_OK on success or error code otherwise.

esp_err_t esp_ble_mesh_generic_client_get_state (esp_ble_mesh_client_common_param_t *params,
                                                 esp_ble_mesh_generic_client_get_state_t *get_state)
Get the value of Generic Server Model states using the Generic Client Model get messages.

备注: If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_generic_message_opcode_t in esp_ble_mesh_defs.h

参数
- params [in] Pointer to BLE Mesh common client parameters.
- get_state [in] Pointer to generic get message value. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

```
esp_err_t esp_ble_mesh_generic_client_set_state(esp_ble_mesh_client_common_param_t *params,
                                                  esp_ble_mesh_generic_client_set_state_t *set_state)
```

Set the value of Generic Server Model states using the Generic Client Model set messages.

备注: If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_generic_message_opcode_t in esp_ble_mesh_defs.h

参数
- params [in] Pointer to BLE Mesh common client parameters.
- set_state [in] Pointer to generic set message value. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

```
esp_err_t esp_ble_mesh_register_generic_server_callback(esp_ble_mesh_generic_server_cb_t callback)
```

Register BLE Mesh Generic Server Model callback.

参数 callback [in] Pointer to the callback function.

返回 ESP_OK on success or error code otherwise.

Unions

```
union esp_ble_mesh_generic_client_get_state_t
#include <esp_ble_mesh_generic_model_api.h> Generic Client Model get message union.
```

Public Members

```
esp_ble_mesh_gen_user_property_get_t user_property_get
For ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_GET
```

```
esp_ble_mesh_gen_admin_property_get_t admin_property_get
For ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_GET
```

```
esp_ble_mesh_gen_manufacturer_property_get_t manufacturer_property_get
For ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET
```

```
esp_ble_mesh_gen_client_properties_get_t client_properties_get
For ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_GET
```
union esp_ble_mesh_generic_client_set_state_t

#include <esp_ble_mesh_generic_model_api.h> Generic Client Model set message union.

Public Members

esp_ble_mesh_gen_onoff_set_t onoff_set
For ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET & ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK

esp_ble_mesh_gen_level_set_t level_set
For ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET & ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET_UNACK

esp_ble_mesh_gen_delta_set_t delta_set
For ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET & ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET_UNACK

esp_ble_mesh_gen_move_set_t move_set
For ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET & ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET_UNACK

esp_ble_mesh_gen_def_trans_time_set_t def_trans_time_set
For ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET & ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET_UNACK

esp_ble_mesh_gen_onpowerup_set_t power_set
For ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET & ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET_UNACK

esp_ble_mesh_gen_power_level_set_t power_level_set
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET & ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET_UNACK

esp_ble_mesh_gen_power_default_set_t power_default_set
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET_UNACK

esp_ble_mesh_gen_power_range_set_t power_range_set
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET & ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET_UNACK

esp_ble_mesh_gen_loc_global_set_t loc_global_set
For ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET & ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET_UNACK

esp_ble_mesh_gen_loc_local_set_t loc_local_set
For ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET & ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET_UNACK

esp_ble_mesh_gen_user_property_set_t user_property_set
For ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET & ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET_UNACK


```c
union esp_ble_mesh_gen_client_status_cb_t

#include <esp_ble_mesh_generic_model_api.h> // Generic Client Model received message union.

Public Members

esp_ble_mesh_gen_onoff_status_cb_t onoff_status
  For ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_STATUS

esp_ble_mesh_gen_level_status_cb_t level_status
  For ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_STATUS

esp_ble_mesh_gen_def_trans_time_status_cb_t def_trans_time_status
  For ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_STATUS

esp_ble_mesh_gen_onpowerup_status_cb_t onpowerup_status
  For ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_STATUS

esp_ble_mesh_gen_power_level_status_cb_t power_level_status
  For ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_STATUS

esp_ble_mesh_gen_power_last_status_cb_t power_last_status
  For ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_STATUS

esp_ble_mesh_gen_power_default_status_cb_t power_default_status
  For ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_STATUS

esp_ble_mesh_gen_power_range_status_cb_t power_range_status
  For ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_STATUS

esp_ble_mesh_gen_battery_status_cb_t battery_status
  For ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_STATUS

esp_ble_mesh_gen_loc_global_status_cb_t location_global_status
  For ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_STATUS

esp_ble_mesh_gen_loc_local_status_cb_t location_local_status
  ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_STATUS

esp_ble_mesh_gen_user_properties_status_cb_t user_properties_status
  ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_STATUS
```
esp_ble_mesh_gen_user_property_status_cb_t user_property_status
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_STATUS

esp_ble_mesh_gen_admin_properties_status_cb_t admin_properties_status
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_STATUS

esp_ble_mesh_gen_admin_property_status_cb_t admin_property_status
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_STATUS

esp_ble_mesh_gen_manufacturer_properties_status_cb_t manufacturer_properties_status
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_STATUS

esp_ble_mesh_gen_manufacturer_property_status_cb_t manufacturer_property_status
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_STATUS

esp_ble_mesh_gen_client_properties_status_cb_t client_properties_status
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_STATUS

union esp_ble_mesh_generic_server_state_change_t
#include <esp_ble_mesh_generic_model_api.h> Generic Server Model state change value union.

**Public Members**

esp_ble_mesh_state_change_gen_onoff_set_t onoff_set
The recv_op in ctx can be used to decide which state is changed. Generic OnOff Set

esp_ble_mesh_state_change_gen_level_set_t level_set
Generic Level Set

esp_ble_mesh_state_change_gen_delta_set_t delta_set
Generic Delta Set

esp_ble_mesh_state_change_gen_move_set_t move_set
Generic Move Set

esp_ble_mesh_state_change_gen_def_trans_time_set_t def_trans_time_set
Generic Default Transition Time Set

esp_ble_mesh_state_change_gen_onpowerup_set_t onpowerup_set
Generic OnPowerUp Set

esp_ble_mesh_state_change_gen_power_level_set_t power_level_set
Generic Power Level Set

esp_ble_mesh_state_change_gen_power_default_set_t power_default_set
Generic Power Default Set
```c
union esp_ble_mesh_state_change_gen_power_range_set_t
    power_range_set
    Generic Power Range Set

union esp_ble_mesh_state_change_gen_loc_global_set_t
    loc_global_set
    Generic Location Global Set

union esp_ble_mesh_state_change_gen_loc_local_set_t
    loc_local_set
    Generic Location Local Set

union esp_ble_mesh_state_change_gen_user_property_set_t
    user_property_set
    Generic User Property Set

union esp_ble_mesh_state_change_gen_admin_property_set_t
    admin_property_set
    Generic Admin Property Set

union esp_ble_mesh_state_change_gen_manu_property_set_t
    manu_property_set
    Generic Manufacturer Property Set
```

```c
union esp_ble_mesh_generic_server_recv_get_msg_t
    #include <esp_ble_mesh_generic_model_api.h> Generic Server Model received get message union.
```

**Public Members**

```c
union esp_ble_mesh_server_recv_gen_user_property_get_t
    user_property
    Generic User Property Get

union esp_ble_mesh_server_recv_gen_admin_property_get_t
    admin_property
    Generic Admin Property Get

union esp_ble_mesh_server_recv_gen_manufacturer_property_get_t
    manu_property
    Generic Manufacturer Property Get

union esp_ble_mesh_server_recv_gen_client_properties_get_t
    client_properties
    Generic Client Properties Get
```

```c
union esp_ble_mesh_generic_server_recv_set_msg_t
    #include <esp_ble_mesh_generic_model_api.h> Generic Server Model received set message union.
```

**Public Members**

```c
union esp_ble_mesh_server_recv_gen_onoff_set_t
    onoff
    Generic OnOff Set/Generic OnOff Set Unack

union esp_ble_mesh_server_recv_gen_level_set_t
    level
    Generic Level Set/Generic Level Set Unack
```


```
void esp_ble_mesh_server_recv_gen_delta_set_t delta
    Generic Delta Set/Generic Delta Set Unack

void esp_ble_mesh_server_recv_gen_move_set_t move
    Generic Move Set/Generic Move Set Unack

void esp_ble_mesh_server_recv_gen_def_trans_time_set_t def_trans_time
    Generic Default Transition Time Set/Generic Default Transition Time Set Unack

void esp_ble_mesh_server_recv_gen_onpowerup_set_t onpowerup
    Generic OnPowerUp Set/Generic OnPowerUp Set Unack

void esp_ble_mesh_server_recv_gen_power_level_set_t power_level
    Generic Power Level Set/Generic Power Level Set Unack

void esp_ble_mesh_server_recv_gen_power_default_set_t power_default
    Generic Power Default Set/Generic Power Default Set Unack

void esp_ble_mesh_server_recv_gen_power_range_set_t power_range
    Generic Power Range Set/Generic Power Range Set Unack

void esp_ble_mesh_server_recv_gen_loc_global_set_t location_global
    Generic Location Global Set/Generic Location Global Set Unack

void esp_ble_mesh_server_recv_gen_loc_local_set_t location_local
    Generic Location Local Set/Generic Location Local Set Unack

void esp_ble_mesh_server_recv_gen_user_property_set_t user_property
    Generic User Property Set/Generic User Property Set Unack

void esp_ble_mesh_server_recv_gen_admin_property_set_t admin_property
    Generic Admin Property Set/Generic Admin Property Set Unack

void esp_ble_mesh_server_recv_gen_manufacturer_property_set_t manu_property
    Generic Manufacturer Property Set/Generic Manufacturer Property Set Unack

union esp_ble_mesh_generic_server_cb_value_t
    #include <esp_ble_mesh_generic_model_api.h> Generic Server Model callback value union.

Public Members

void esp_ble_mesh_server_state_change_t state_change
    ESP_BLE_MESH_GENERIC_SERVER_STATE_CHANGE_EVT

void esp_ble_mesh_server_recv_msg_t get
    ESP_BLE_MESH_GENERIC_SERVER_RECV_MSG_EVT

void esp_ble_mesh_server_recv_set_msg_t set
    ESP_BLE_MESH_GENERIC_SERVER_RECV_SET_MSG_EVT
```

**Structures**

**struct esp_ble_mesh_gen_onoff_set_t**
Bluetooth Mesh Generic Client Model Get and Set parameters structure.
Parameters of Generic OnOff Set.

**Public Members**

- `bool op_en`
  Indicate if optional parameters are included
- `uint8_t onoff`
  Target value of Generic OnOff state
- `uint8_t tid`
  Transaction ID
- `uint8_t trans_time`
  Time to complete state transition (optional)
- `uint8_t delay`
  Indicate message execution delay (C.1)

**struct esp_ble_mesh_gen_level_set_t**
Parameters of Generic Level Set.

**Public Members**

- `bool op_en`
  Indicate if optional parameters are included
- `int16_t level`
  Target value of Generic Level state
- `uint8_t tid`
  Transaction ID
- `uint8_t trans_time`
  Time to complete state transition (optional)
- `uint8_t delay`
  Indicate message execution delay (C.1)

**struct esp_ble_mesh_gen_delta_set_t**
Parameters of Generic Delta Set.
Public Members

`bool op_en`
   Indicate if optional parameters are included

`int32_t level`
   Delta change of Generic Level state

`uint8_t tid`
   Transaction ID

`uint8_t trans_time`
   Time to complete state transition (optional)

`uint8_t delay`
   Indicate message execution delay (C.1)

`struct esp_ble_mesh_gen_move_set_t`
   Parameters of Generic Move Set.

Public Members

`bool op_en`
   Indicate if optional parameters are included

`int16_t delta_level`
   Delta Level step to calculate Move speed for Generic Level state

`uint8_t tid`
   Transaction ID

`uint8_t trans_time`
   Time to complete state transition (optional)

`uint8_t delay`
   Indicate message execution delay (C.1)

`struct esp_ble_mesh_gen_def_trans_time_set_t`
   Parameter of Generic Default Transition Time Set.

Public Members

`uint8_t trans_time`
   The value of the Generic Default Transition Time state

`struct esp_ble_mesh_gen_onpowerup_set_t`
   Parameter of Generic OnPowerUp Set.
Public Members

`uint8_t onpowerup`
The value of the Generic OnPowerUp state

`struct esp_ble_mesh_gen_power_level_set_t`
Parameters of Generic Power Level Set.

Public Members

`bool op_en`
Indicate if optional parameters are included

`uint16_t power`
Target value of Generic Power Actual state

`uint8_t tid`
Transaction ID

`uint8_t trans_time`
Time to complete state transition (optional)

`uint8_t delay`
Indicate message execution delay (C.1)

`struct esp_ble_mesh_gen_power_default_set_t`
Parameter of Generic Power Default Set.

Public Members

`uint16_t power`
The value of the Generic Power Default state

`struct esp_ble_mesh_gen_power_range_set_t`
Parameters of Generic Power Range Set.

Public Members

`uint16_t range_min`
Value of Range Min field of Generic Power Range state

`uint16_t range_max`
Value of Range Max field of Generic Power Range state

`struct esp_ble_mesh_gen_loc_global_set_t`
Parameters of Generic Location Global Set.
Public Members

```c
int32_t global_latitude
Global Coordinates (Latitude)
```

```c
int32_t global_longitude
Global Coordinates (Longitude)
```

```c
int16_t global_altitude
Global Altitude
```

```c
struct esp_ble_mesh_gen_loc_local_set_t
Parameters of Generic Location Local Set.
```

Public Members

```c
int16_t local_north
Local Coordinates (North)
```

```c
int16_t local_east
Local Coordinates (East)
```

```c
int16_t local_altitude
Local Altitude
```

```c
uint8_t floor_number
Floor Number
```

```c
uint16_t uncertainty
Uncertainty
```

```c
struct esp_ble_mesh_gen_user_property_get_t
Parameter of Generic User Property Get.
```

Public Members

```c
uint16_t property_id
Property ID identifying a Generic User Property
```

```c
struct esp_ble_mesh_gen_user_property_set_t
Parameters of Generic User Property Set.
```

Public Members

```c
uint16_t property_id
Property ID identifying a Generic User Property
```
struct net_buf_simple *property_value
    Raw value for the User Property

struct esp_ble_mesh_gen_admin_property_get_t
    Parameter of Generic Admin Property Get.

    **Public Members**

    uint16_t property_id
        Property ID identifying a Generic Admin Property

    struct esp_ble_mesh_gen_admin_property_set_t
        Parameters of Generic Admin Property Set.

    **Public Members**

    uint16_t property_id
        Property ID identifying a Generic Admin Property

    uint8_t user_access
        Enumeration indicating user access

    struct net_buf_simple *property_value
        Raw value for the Admin Property

struct esp_ble_mesh_gen_manufacturer_property_get_t
    Parameter of Generic Manufacturer Property Get.

    **Public Members**

    uint16_t property_id
        Property ID identifying a Generic Manufacturer Property

struct esp_ble_mesh_gen_manufacturer_property_set_t
    Parameters of Generic Manufacturer Property Set.

    **Public Members**

    uint16_t property_id
        Property ID identifying a Generic Manufacturer Property

    uint8_t user_access
        Enumeration indicating user access

struct esp_ble_mesh_gen_client_properties_get_t
    Parameter of Generic Client Properties Get.
**Public Members**

```c
uint16_t property_id
```
A starting Client Property ID present within an element

```c
struct esp_ble_mesh_gen_onoff_status_cb_t
```
Bluetooth Mesh Generic Client Model Get and Set callback parameters structure.
Parameters of Generic OnOff Status.

**Public Members**

```c
bool op_en
```
Indicate if optional parameters are included

```c
uint8_t present_onoff
```
Current value of Generic OnOff state

```c
uint8_t target_onoff
```
Target value of Generic OnOff state (optional)

```c
uint8_t remain_time
```
Time to complete state transition (C.1)

**Public Members**

```c
struct esp_ble_mesh_gen_level_status_cb_t
```
Parameters of Generic Level Status.

**Public Members**

```c
bool op_en
```
Indicate if optional parameters are included

```c
int16_t present_level
```
Current value of Generic Level state

```c
int16_t target_level
```
Target value of the Generic Level state (optional)

```c
uint8_t remain_time
```
Time to complete state transition (C.1)

**Public Members**

```c
struct esp_ble_mesh_gen_def_trans_time_status_cb_t
```
Parameter of Generic Default Transition Time Status.
uint8_t trans_time
  The value of the Generic Default Transition Time state

struct esp_ble_mesh_gen_onpowerup_status_cb_t

Public Members

uint8_t onpowerup
  The value of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_level_status_cb_t
  Parameters of Generic Power Level Status.

Public Members

bool op_en
  Indicate if optional parameters are included

uint16_t present_power
  Current value of Generic Power Actual state

uint16_t target_power
  Target value of Generic Power Actual state (optional)

uint8_t remain_time
  Time to complete state transition (C.1)

struct esp_ble_mesh_gen_power_last_status_cb_t
  Parameter of Generic Power Last Status.

Public Members

uint16_t power
  The value of the Generic Power Last state

struct esp_ble_mesh_gen_power_default_status_cb_t
  Parameter of Generic Power Default Status.

Public Members

uint16_t power
  The value of the Generic Default Last state

struct esp_ble_mesh_gen_power_range_status_cb_t
  Parameters of Generic Power Range Status.
Public Members

uint8_t status_code
Status Code for the request message

uint16_t range_min
Value of Range Min field of Generic Power Range state

uint16_t range_max
Value of Range Max field of Generic Power Range state

struct esp_ble_mesh_gen_battery_status_cb_t
Parameters of Generic Battery Status.

Public Members

uint32_t battery_level
Value of Generic Battery Level state

uint32_t time_to_discharge
Value of Generic Battery Time to Discharge state

uint32_t time_to_charge
Value of Generic Battery Time to Charge state

uint32_t flags
Value of Generic Battery Flags state

struct esp_ble_mesh_gen_loc_global_status_cb_t
Parameters of Generic Location Global Status.

Public Members

int32_t global_latitude
Global Coordinates (Latitude)

int32_t global_longitude
Global Coordinates (Longitude)

int16_t global_altitude
Global Altitude

struct esp_ble_mesh_gen_loc_local_status_cb_t
Parameters of Generic Location Local Status.
**Public Members**

int16_t `local_north`
Local Coordinates (North)

int16_t `local_east`
Local Coordinates (East)

int16_t `local_altitude`
Local Altitude

uint8_t `floor_number`
Floor Number

uint16_t `uncertainty`
Uncertainty

```c
struct esp_ble_mesh_gen_user_properties_status_cb_t
Parameter of Generic User Properties Status.
```

```c
struct net_buf_simple *property_ids
Buffer contains a sequence of N User Property IDs
```

```c
struct esp_ble_mesh_gen_user_property_status_cb_t
Parameters of Generic User Property Status.
```

**Public Members**

```c
bool op_en
Indicate if optional parameters are included
```

```c
uint16_t property_id
Property ID identifying a Generic User Property
```

```c
uint8_t user_access
Enumeration indicating user access (optional)
```

```c
struct net_buf_simple *property_value
Raw value for the User Property (C.1)
```

```c
struct esp_ble_mesh_gen_admin_properties_status_cb_t
Parameter of Generic Admin Properties Status.
```
Public Members

struct net_buf_simple *property_ids
    Buffer contains a sequence of N Admin Property IDs

struct esp_ble_mesh_gen_admin_property_status_cb_t
    Parameters of Generic Admin Property Status.

Public Members

bool op_en
    Indicate if optional parameters are included

uint16_t property_id
    Property ID identifying a Generic Admin Property

uint8_t user_access
    Enumeration indicating user access (optional)

struct net_buf_simple *property_value
    Raw value for the Admin Property (C.1)

struct esp_ble_mesh_gen_manufacturer_properties_status_cb_t
    Parameter of Generic Manufacturer Properties Status.

Public Members

struct net_buf_simple *property_ids
    Buffer contains a sequence of N Manufacturer Property IDs

struct esp_ble_mesh_gen_manufacturer_property_status_cb_t
    Parameters of Generic Manufacturer Property Status.

Public Members

bool op_en
    Indicate if optional parameters are included

uint16_t property_id
    Property ID identifying a Generic Manufacturer Property

uint8_t user_access
    Enumeration indicating user access (optional)

struct net_buf_simple *property_value
    Raw value for the Manufacturer Property (C.1)
struct `esp_ble_mesh_gen_client_properties_status_cb_t`
Parameter of Generic Client Properties Status.

**Public Members**

```
struct net_buf_simple *property_ids
```
Buffer contains a sequence of N Client Property IDs

struct `esp_ble_mesh_generic_client_cb_param_t`
Generic Client Model callback parameters

**Public Members**

```
int error_code
```
Appropriate error code

```
esp_ble_mesh_client_common_param_t *params
```
The client common parameters.

```
esp_ble_mesh_gen_client_status_cb_t status_cb
```
The generic status message callback values

struct `esp_ble_mesh_gen_onoff_state_t`
Parameters of Generic OnOff state

**Public Members**

```
uint8_t onoff
```
The present value of the Generic OnOff state

```
uint8_t target_onoff
```
The target value of the Generic OnOff state

struct `esp_ble_mesh_gen_onoff_srv_t`
User data of Generic OnOff Server Model

**Public Members**

```
esp_ble_mesh_model_t *model
```
Pointer to the Generic OnOff Server Model. Initialized internally.

```
esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl
```
Response control of the server model received messages

```
esp_ble_mesh_gen_onoff_state_t state
```
Parameters of the Generic OnOff state
**Chapter 2. API**

*esp_ble_mesh_last_msg_info_t* `last`
Parameters of the last received set message

*esp_ble_mesh_state_transition_t* `transition`
Parameters of state transition

**struct esp_ble_mesh_gen_level_state_t**
Parameters of Generic Level state

**Public Members**

`int16_t level`
The present value of the Generic Level state

`int16_t target_level`
The target value of the Generic Level state

`int16_t last_level`
When a new transaction starts, level should be set to last_last, and use “level + incoming delta” to calculate the target level. In another word, “last_level” is used to record “level” of the last transaction, and “last_delta” is used to record the previously received delta_level value. The last value of the Generic Level state

`int32_t last_delta`
The last delta change of the Generic Level state

`bool move_start`
Indicate if the transition of the Generic Level state has been started

`bool positive`
Indicate if the transition is positive or negative

**struct esp_ble_mesh_gen_level_srv_t**
User data of Generic Level Server Model

**Public Members**

`esp_ble_mesh_model_t *model`
Pointer to the Generic Level Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl`
Response control of the server model received messages

`esp_ble_mesh_gen_level_state_t state`
Parameters of the Generic Level state

`esp_ble_mesh_last_msg_info_t last`
Parameters of the last received set message
`esp_ble_mesh_state_transition_t` 
Parameters of state transition

`int32_t tt_delta_level`
Delta change value of level state transition

`struct esp_ble_mesh_gen_def_trans_time_state_t`
Parameters of Generic Default Transition Time state

**Public Members**

`uint8_t trans_time`
The value of the Generic Default Transition Time state

`struct esp_ble_mesh_gen_def_trans_time_srv_t`
User data of Generic Default Transition Time Server Model

**Public Members**

`esp_ble_mesh_model_t *model`
Pointer to the Generic Default Transition Time Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`esp_ble_mesh_gen_def_trans_time_state_t state`
Parameters of the Generic Default Transition Time state

`struct esp_ble_mesh_gen_onpowerup_state_t`
Parameter of Generic OnPowerUp state

**Public Members**

`uint8_t onpowerup`
The value of the Generic OnPowerUp state

`struct esp_ble_mesh_gen_power_onoff_srv_t`
User data of Generic Power OnOff Server Model

**Public Members**

`esp_ble_mesh_model_t *model`
Pointer to the Generic Power OnOff Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages
Chapter 2. API

.esp_ble_mesh_gen_onpowerup_state_t *state
Parameters of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_onoff_setup_srv_t
User data of Generic Power OnOff Setup Server Model

Public Members

.esp_ble_mesh_model_t *model
Point to the Generic Power OnOff Setup Server Model. Initialized internally.

.esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl
Response control of the server model received messages

.esp_ble_mesh_gen_onpowerup_state_t *state
Parameters of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_level_state_t
Parameters of Generic Power Level state

Public Members

uint16_t power_actual
The present value of the Generic Power Actual state

uint16_t target_power_actual
The target value of the Generic Power Actual state

uint16_t power_last
The value of the Generic Power Last state

uint16_t power_default
The value of the Generic Power Default state

uint8_t status_code
The status code of setting Generic Power Range state

uint16_t power_range_min
The minimum value of the Generic Power Range state

uint16_t power_range_max
The maximum value of the Generic Power Range state

struct esp_ble_mesh_gen_power_level_srv_t
User data of Generic Power Level Server Model
**Public Members**

`esp_ble_mesh_model_t *model`

Pointer to the Generic Power Level Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`

Response control of the server model received messages

`esp_ble_mesh_gen_power_level_state_t *state`

Parameters of the Generic Power Level state

`esp_ble_mesh_last_msg_info_t last`

Parameters of the last received set message

`esp_ble_mesh_state_transition_t transition`

Parameters of state transition

`int32_t ttDelta_level`

Delta change value of level state transition

`struct esp_ble_mesh_gen_power_level_setup_srv_t`  

User data of Generic Power Level Setup Server Model

**Public Members**

`esp_ble_mesh_model_t *model`

Pointer to the Generic Power Level Setup Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`

Response control of the server model received messages

`esp_ble_mesh_gen_power_level_state_t *state`

Parameters of the Generic Power Level state

`struct esp_ble_mesh_gen_battery_state_t`  

Parameters of Generic Battery state

**Public Members**

`uint32_t battery_level`

The value of the Generic Battery Level state

`uint32_t time_to_discharge`

The value of the Generic Battery Time to Discharge state

`uint32_t time_to_charge`

The value of the Generic Battery Time to Charge state
Chapter 2. API

**Public Members**

`uint32_t battery_flags`

The value of the Generic Battery Flags state

`struct esp_ble_mesh_gen_battery_srv_t`

User data of Generic Battery Server Model

```

**Public Members**

`esp_ble_mesh_model_t *model`

Pointer to the Generic Battery Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl`

Response control of the server model received messages

`esp_ble_mesh_gen_battery_state_t state`

Parameters of the Generic Battery state

`struct esp_ble_mesh_gen_location_srv_t`

User data of Generic Location Server Model

**Public Members**

`int32_t global_latitude`

The value of the Global Latitude field

`int32_t global_longitude`

The value of the Global Longitude field

`int16_t global_altitude`

The value of the Global Altitude field

`int16_t local_north`

The value of the Local North field

`int16_t local_east`

The value of the Local East field

`int16_t local_altitude`

The value of the Local Altitude field

`uint8_t floor_number`

The value of the Floor Number field

`uint16_t uncertainty`

The value of the Uncertainty field

`struct esp_ble_mesh_gen_location_state_t`

Parameters of Generic Location state

Espressif Systems 654

Submit Document Feedback
Public Members

* esp_ble_mesh_model_t *model
  Pointer to the Generic Location Server Model. Initialized internally.

* esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl
  Response control of the server model received messages

* esp_ble_mesh_gen_location_state_t *state
  Parameters of the Generic Location state

struct esp_ble_mesh_gen_location_setup_srv_t
  User data of Generic Location Setup Server Model

Public Members

* esp_ble_mesh_model_t *model
  Pointer to the Generic Location Setup Server Model. Initialized internally.

* esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl
  Response control of the server model received messages

* esp_ble_mesh_gen_location_state_t *state
  Parameters of the Generic Location state

struct esp_ble_mesh_generic_property_t
  Parameters of Generic Property states

Public Members

uint16_t id
  The value of User/Admin/Manufacturer Property ID

uint8_t user_access
  The value of User Access field

uint8_t admin_access
  The value of Admin Access field

uint8_t manu_access
  The value of Manufacturer Access field

struct net_buf_simple *val
  The value of User/Admin/Manufacturer Property

struct esp_ble_mesh_gen_user_prop_srv_t
  User data of Generic User Property Server Model
Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Generic User Property Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`uint8_t property_count`
Generic User Property count

`esp_ble_mesh_generic_property_t *properties`
Parameters of the Generic User Property state

```c
struct esp_ble_mesh_gen_admin_prop_srv_t
User data of Generic Admin Property Server Model
```

Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Generic Admin Property Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`uint8_t property_count`
Generic Admin Property count

`esp_ble_mesh_generic_property_t *properties`
Parameters of the Generic Admin Property state

```c
struct esp_ble_mesh_gen_manu_prop_srv_t
User data of Generic Manufacturer Property Server Model
```

Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Generic Manufacturer Property Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`uint8_t property_count`
Generic Manufacturer Property count

`esp_ble_mesh_generic_property_t *properties`
Parameters of the Generic Manufacturer Property state
struct `esp_ble_mesh_gen_client_prop_srv_t`  
User data of Generic Client Property Server Model

**Public Members**

`esp_ble_mesh_model_t *model`
Pointer to the Generic Client Property Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t * rsp_ctrl`
Response control of the server model received messages

`uint8_t id_count`
Generic Client Property ID count

`uint16_t *property_ids`
Parameters of the Generic Client Property state

struct `esp_ble_mesh_state_change_gen_onoff_set_t`  
Parameter of Generic OnOff Set state change event

**Public Members**

`uint8_t onoff`
The value of Generic OnOff state

struct `esp_ble_mesh_state_change_gen_level_set_t`  
Parameter of Generic Level Set state change event

**Public Members**

`int16_t level`
The value of Generic Level state

struct `esp_ble_mesh_state_change_gen_delta_set_t`  
Parameter of Generic Delta Set state change event

**Public Members**

`int16_t level`
The value of Generic Level state

struct `esp_ble_mesh_state_change_gen_move_set_t`  
Parameter of Generic Move Set state change event
Public Members

int16_t level
    The value of Generic Level state

struct esp_ble_mesh_state_change_gen_def_trans_time_set_t
    Parameter of Generic Default Transition Time Set state change event

Public Members

uint8_t trans_time
    The value of Generic Default Transition Time state

struct esp_ble_mesh_state_change_gen_onpowerup_set_t
    Parameter of Generic OnPowerUp Set state change event

Public Members

uint8_t onpowerup
    The value of Generic OnPowerUp state

struct esp_ble_mesh_state_change_gen_power_level_set_t
    Parameter of Generic Power Level Set state change event

Public Members

uint16_t power
    The value of Generic Power Actual state

struct esp_ble_mesh_state_change_gen_power_default_set_t
    Parameter of Generic Power Default Set state change event

Public Members

uint16_t power
    The value of Generic Power Default state

struct esp_ble_mesh_state_change_gen_power_range_set_t
    Parameters of Generic Power Range Set state change event

Public Members

uint16_t range_min
    The minimum value of Generic Power Range state
uint16_t `range_max`
  The maximum value of Generic Power Range state

`struct esp_ble_mesh_state_change_gen_loc_global_set_t`
Parameters of Generic Location Global Set state change event

**Public Members**

`int32_t latitude`
  The Global Latitude value of Generic Location state

`int32_t longitude`
  The Global Longitude value of Generic Location state

`int16_t altitude`
  The Global Altitude value of Generic Location state

`struct esp_ble_mesh_state_change_gen_loc_local_set_t`
Parameters of Generic Location Local Set state change event

**Public Members**

`int16_t north`
  The Local North value of Generic Location state

`int16_t east`
  The Local East value of Generic Location state

`int16_t altitude`
  The Local Altitude value of Generic Location state

`uint8_t floor_number`
  The Floor Number value of Generic Location state

`uint16_t uncertainty`
  The Uncertainty value of Generic Location state

`struct esp_ble_mesh_state_change_gen_user_property_set_t`
Parameters of Generic User Property Set state change event

**Public Members**

`uint16_t id`
  The property id of Generic User Property state
struct net_buf_simple *value
  The property value of Generic User Property state

struct esp_ble_mesh_state_change_gen_admin_property_set_t
  Parameters of Generic Admin Property Set state change event

Public Members

uint16_t id
  The property id of Generic Admin Property state

uint8_t access
  The property access of Generic Admin Property state

struct net_buf_simple *value
  The property value of Generic Admin Property state

struct esp_ble_mesh_state_change_gen_manu_property_set_t
  Parameters of Generic Manufacturer Property Set state change event

Public Members

uint16_t id
  The property id of Generic Manufacturer Property state

uint8_t access
  The property value of Generic Manufacturer Property state

struct esp_ble_mesh_server_recv_gen_user_property_get_t
  Context of the received Generic User Property Get message

Public Members

uint16_t property_id
  Property ID identifying a Generic User Property

struct esp_ble_mesh_server_recv_gen_admin_property_get_t
  Context of the received Generic Admin Property Get message

Public Members

uint16_t property_id
  Property ID identifying a Generic Admin Property

struct esp_ble_mesh_server_recv_gen_manufacturer_property_get_t
  Context of the received Generic Manufacturer Property message
Chapter 2. API Reference

**Public Members**

**uint16_t** `property_id`
- Property ID identifying a Generic Manufacturer Property

**struct esp_ble_mesh_server_recv_gen_client_properties_get_t**
- Context of the received Generic Client Properties Get message

**Public Members**

**uint16_t** `property_id`
- A starting Client Property ID present within an element

**struct esp_ble_mesh_server_recv_gen_onoff_set_t**
- Context of the received Generic OnOff Set message

**Public Members**

**bool** `op_en`
- Indicate if optional parameters are included

**uint8_t** `onoff`
- Target value of Generic OnOff state

**uint8_t** `tid`
- Transaction ID

**uint8_t** `trans_time`
- Time to complete state transition (optional)

**uint8_t** `delay`
- Indicate message execution delay (C.1)

**struct esp_ble_mesh_server_recv_gen_level_set_t**
- Context of the received Generic Level Set message

**Public Members**

**bool** `op_en`
- Indicate if optional parameters are included

**int16_t** `level`
- Target value of Generic Level state

**uint8_t** `tid`
- Transaction ID
### Public Members

- **bool op_en**
  - Indicate if optional parameters are included

- **int32_t delta_level**
  - Delta change of Generic Level state

- **uint8_t tid**
  - Transaction ID

- **uint8_t trans_time**
  - Time to complete state transition (optional)

- **uint8_t delay**
  - Indicate message execution delay (C.1)

### Context of the received Generic Delta Set message

- **struct esp_ble_mesh_server_recv_gen_delta_set_t**

### Public Members

- **bool op_en**
  - Indicate if optional parameters are included

- **int16_t delta_level**
  - Delta Level step to calculate Move speed for Generic Level state

- **uint8_t tid**
  - Transaction ID

- **uint8_t trans_time**
  - Time to complete state transition (optional)

- **uint8_t delay**
  - Indicate message execution delay (C.1)

### Context of the received Generic Move Set message

- **struct esp_ble_mesh_server_recv_gen_move_set_t**

### Context of the received Generic Default Transition Time Set message

- **struct esp_ble_mesh_server_recv_gen_def_trans_time_set_t**

---

uint8_t \texttt{trans\_time}

Time to complete state transition (optional)

uint8_t \texttt{delay}

Indicate message execution delay (C.1)

struct \texttt{esp\_ble\_mesh\_server\_recv\_gen\_delta\_set\_t}

Context of the received Generic Delta Set message

Public Members

bool \texttt{op\_en}

Indicate if optional parameters are included

int32_t \texttt{delta\_level}

Delta change of Generic Level state

uint8_t \texttt{tid}

Transaction ID

uint8_t \texttt{trans\_time}

Time to complete state transition (optional)

uint8_t \texttt{delay}

Indicate message execution delay (C.1)

struct \texttt{esp\_ble\_mesh\_server\_recv\_gen\_move\_set\_t}

Context of the received Generic Move Set message

Public Members

bool \texttt{op\_en}

Indicate if optional parameters are included

int16_t \texttt{delta\_level}

Delta Level step to calculate Move speed for Generic Level state

uint8_t \texttt{tid}

Transaction ID

uint8_t \texttt{trans\_time}

Time to complete state transition (optional)

uint8_t \texttt{delay}

Indicate message execution delay (C.1)

struct \texttt{esp\_ble\_mesh\_server\_recv\_gen\_def\_trans\_time\_set\_t}

Context of the received Generic Default Transition Time Set message
Public Members

uint8_t trans_time
The value of the Generic Default Transition Time state

struct esp_ble_mesh_server_recv_gen_onpowerup_set_t
Context of the received Generic OnPowerUp Set message

Public Members

uint8_t onpowerup
The value of the Generic OnPowerUp state

struct esp_ble_mesh_server_recv_gen_power_level_set_t
Context of the received Generic Power Level Set message

Public Members

bool op_en
Indicate if optional parameters are included

uint16_t power
Target value of Generic Power Actual state

uint8_t tid
Transaction ID

uint8_t trans_time
Time to complete state transition (optional)

uint8_t delay
Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_gen_power_default_set_t
Context of the received Generic Power Default Set message

Public Members

uint16_t power
The value of the Generic Power Default state

struct esp_ble_mesh_server_recv_gen_power_range_set_t
Context of the received Generic Power Range Set message
**Chapter 2. API 参考**

**Public Members**

```c
uint16_t range_min
  Value of Range Min field of Generic Power Range state
```

```c
uint16_t range_max
  Value of Range Max field of Generic Power Range state
```

```c
struct esp_ble_mesh_server_recv_gen_loc_global_set_t
  Context of the received Generic Location Global Set message
```

**Public Members**

```c
int32_t global_latitude
  Global Coordinates (Latitude)
```

```c
int32_t global_longitude
  Global Coordinates (Longitude)
```

```c
int16_t global_altitude
  Global Altitude
```

```c
struct esp_ble_mesh_server_recv_gen_loc_local_set_t
  Context of the received Generic Location Local Set message
```

**Public Members**

```c
int16_t local_north
  Local Coordinates (North)
```

```c
int16_t local_east
  Local Coordinates (East)
```

```c
int16_t local_altitude
  Local Altitude
```

```c
uint8_t floor_number
  Floor Number
```

```c
uint16_t uncertainty
  Uncertainty
```

```c
struct esp_ble_mesh_server_recv_gen_user_property_set_t
  Context of the received Generic User Property Set message
```
Public Members

`uint16_t property_id`
Property ID identifying a Generic User Property

`struct net_buf_simple *property_value`
Raw value for the User Property

`struct esp_ble_mesh_server_recv_gen_admin_property_set_t`
Context of the received Generic Admin Property Set message

Public Members

`uint16_t property_id`
Property ID identifying a Generic Admin Property

`uint8_t user_access`
Enumeration indicating user access

`struct net_buf_simple *property_value`
Raw value for the Admin Property

`struct esp_ble_mesh_server_recv_gen_manufacturer_property_set_t`
Context of the received Generic Manufacturer Property Set message

Public Members

`uint16_t property_id`
Property ID identifying a Generic Manufacturer Property

`uint8_t user_access`
Enumeration indicating user access

`struct esp_ble_mesh_generic_server_cb_param_t`
Generic Server Model callback parameters

Public Members

`esp_ble_mesh_model_t *model`
Pointer to Generic Server Models

`esp_ble_mesh_msg_ctx_t *ctx`
Context of the received messages

`esp_ble_mesh_generic_server_cb_value_t value`
Value of the received Generic Messages
** Macros **

** ESP_BLE_MESH_MODEL_GEN_ONOFF_CLI (cli_pub, cli_data) **
Define a new Generic OnOff Client Model.

备注:  This API needs to be called for each element on which the application needs to have a Generic OnOff Client Model.

参数
- cli_pub – Pointer to the unique struct esp_ble_mesh_model_pub_t.
- cli_data – Pointer to the unique struct esp_ble_mesh_client_t.

返回 New Generic OnOff Client Model instance.

** ESP_BLE_MESH_MODEL_GEN_LEVEL_CLI (cli_pub, cli_data) **
Define a new Generic Level Client Model.

备注:  This API needs to be called for each element on which the application needs to have a Generic Level Client Model.

参数
- cli_pub – Pointer to the unique struct esp_ble_mesh_model_pub_t.
- cli_data – Pointer to the unique struct esp_ble_mesh_client_t.

返回 New Generic Level Client Model instance.

** ESP_BLE_MESH_MODEL_GEN_DEF_TRANS_TIME_CLI (cli_pub, cli_data) **
Define a new Generic Default Transition Time Client Model.

备注:  This API needs to be called for each element on which the application needs to have a Generic Default Transition Time Client Model.

参数
- cli_pub – Pointer to the unique struct esp_ble_mesh_model_pub_t.
- cli_data – Pointer to the unique struct esp_ble_mesh_client_t.

返回 New Generic Default Transition Time Client Model instance.

** ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_CLI (cli_pub, cli_data) **
Define a new Generic Power OnOff Client Model.

备注:  This API needs to be called for each element on which the application needs to have a Generic Power OnOff Client Model.

参数
- cli_pub – Pointer to the unique struct esp_ble_mesh_model_pub_t.
- cli_data – Pointer to the unique struct esp_ble_mesh_client_t.

返回 New Generic Power OnOff Client Model instance.

** ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_CLI (cli_pub, cli_data) **
Define a new Generic Power Level Client Model.

备注:  This API needs to be called for each element on which the application needs to have a Generic Power Level Client Model.
Chapter 2. API

**Define a new Generic Battery Client Model.**

```c
ESP_BLE_MESH_MODEL_GEN_BATTERY_CLI(cli_pub, cli_data)
```

**Define a new Generic Location Client Model.**

```c
ESP_BLE_MESH_MODEL_GEN_LOCATION_CLI(cli_pub, cli_data)
```

**Define a new Generic Property Client Model.**

```c
ESP_BLE_MESH_MODEL_GEN_PROPERTY_CLI(cli_pub, cli_data)
```

**Define a new Generic OnOff Server Model.**

```c
ESP_BLE_MESH_MODEL_GEN_ONOFF_SRV(srv_pub, srv_data)
```

---

**Parameter**

- `cli_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `cli_data` - Pointer to the unique struct `esp_ble_mesh_client_t`.

**Return**

New Generic Power Level Client Model instance.

**备注:** This API needs to be called for each element on which the application needs to have a Generic Battery Client Model.

---

**Parameter**

- `cli_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `cli_data` - Pointer to the unique struct `esp_ble_mesh_client_t`.

**Return**

New Generic Battery Client Model instance.

**备注:** This API needs to be called for each element on which the application needs to have a Generic Location Client Model.

---

**Parameter**

- `cli_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `cli_data` - Pointer to the unique struct `esp_ble_mesh_client_t`.

**Return**

New Generic Location Client Model instance.

**备注:** This API needs to be called for each element on which the application needs to have a Generic Property Client Model.

---

**Parameter**

- `cli_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `cli_data` - Pointer to the unique struct `esp_ble_mesh_client_t`.

**Return**

New Generic Location Client Model instance.

**备注:** This API needs to be called for each element on which the application needs to have a Generic OnOff Server Model.

1. The Generic OnOff Server Model is a root model.
   a. This model shall support model publication and model subscription.

---

**Parameter**

- `srv_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_gen_onoff_srv_t`.

**Return**

New Generic OnOff Server Model instance.
Chapter 2. API 参考

**ESP_BLE_MESH_MODEL_GEN_LEVEL_SRV** (srv_pub, srv_data)
Define a new Generic Level Server Model.

| 备注 | 1. The Generic Level Server Model is a root model.
| a. This model shall support model publication and model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_level_srv_t`.

**返回** New Generic Level Server Model instance.

**ESP_BLE_MESH_MODEL_GEN_DEF_TRANS_TIME_SRV** (srv_pub, srv_data)
Define a new Generic Default Transition Time Server Model.

| 备注 | 1. The Generic Default Transition Time Server Model is a root model.
| a. This model shall support model publication and model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_def_trans_time_srv_t`.

**返回** New Generic Default Transition Time Server Model instance.

**ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_SRV** (srv_pub, srv_data)
Define a new Generic Power OnOff Server Model.

| 备注 | 1. The Generic Power OnOff Server model extends the Generic OnOff Server model. When this model is present on an element, the corresponding Generic Power OnOff Setup Server model shall also be present.
| a. This model may be used to represent a variety of devices that do not fit any of the model descriptions that have been defined but support the generic properties of On/Off.
| b. This model shall support model publication and model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_power_onoff_srv_t`.

**返回** New Generic Power OnOff Server Model instance.

**ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_SETUP_SRV** (srv_pub, srv_data)
Define a new Generic Power OnOff Setup Server Model.

| a. This model shall support model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_power_onoff_setup_srv_t`.

**返回** New Generic Power OnOff Setup Server Model instance.
ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_SRV (srv_pub, srv_data)
Define a new Generic Power Level Server Model.

### Parameters
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_power_level_srv_t`.

### Return
New Generic Power Level Server Model instance.

ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_SETUP_SRV (srv_pub, srv_data)
Define a new Generic Power Level Setup Server Model.

### Parameters
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_power_level_setup_srv_t`.

### Return
New Generic Power Level Setup Server Model instance.

ESP_BLE_MESH_MODEL_GEN_BATTERY_SRV (srv_pub, srv_data)
Define a new Generic Battery Server Model.

### Parameters
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_battery_srv_t`.

### Return
New Generic Battery Server Model instance.

ESP_BLE_MESH_MODEL_GEN_LOCATION_SRV (srv_pub, srv_data)
Define a new Generic Location Server Model.

### Parameters
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_location_srv_t`.

### Return
New Generic Location Server Model instance.
### ESP_BLE_MESH_MODEL_GEN_LOCATION_SETUP_SRV (srv_pub, srv_data)
Define a new Generic Location Setup Server Model.

**备注:** 1. The Generic Location Setup Server model extends the Generic Location Server model.

   a. This model shall support model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_location_setup_srv_t`.

**返回** New Generic Location Setup Server Model instance.

### ESP_BLE_MESH_MODEL_GEN_USER_PROP_SRV (srv_pub, srv_data)
Define a new Generic User Property Server Model.

**备注:** 1. The Generic User Property Server model is a root model.

   a. This model shall support model publication and model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_user_prop_srv_t`.

**返回** New Generic User Property Server Model instance.

### ESP_BLE_MESH_MODEL_GEN_ADMIN_PROP_SRV (srv_pub, srv_data)
Define a new Generic Admin Property Server Model.

**备注:** 1. The Generic Admin Property Server model extends the Generic User Property Server model.

   a. This model shall support model publication and model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_admin_prop_srv_t`.

**返回** New Generic Admin Property Server Model instance.

### ESP_BLE_MESH_MODEL_GEN_MANUFACTURER_PROP_SRV (srv_pub, srv_data)
Define a new Generic Manufacturer Property Server Model.

**备注:** 1. The Generic Manufacturer Property Server model extends the Generic User Property Server model.

   a. This model shall support model publication and model subscription.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_gen_manu_prop_srv_t`.

**返回** New Generic Manufacturer Property Server Model instance.

### ESP_BLE_MESH_MODEL_GEN_CLIENT_PROP_SRV (srv_pub, srv_data)
Define a new Generic Client Property Server Model.

**备注:** 1. The Generic Client Property Server model is a root model.
This model shall support model publication and model subscription.

**Parameters**
- **srv_pub** - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** - Pointer to the unique struct `esp_ble_mesh_gen_client_prop_srv_t`.

New Generic Client Property Server Model instance.

**Type Definitions**

```c
typedef void (*esp_ble_mesh_generic_client_cb_t)(esp_ble_mesh_generic_client_cb_event_t event,
                                                esp_ble_mesh_generic_client_cb_param_t *param);
```

Bluetooth Mesh Generic Client Model function.
Generic Client Model callback function type
- **Param event** Event type
- **Param param** Pointer to callback parameter

```c
typedef void (*esp_ble_mesh_generic_server_cb_t)(esp_ble_mesh_generic_server_cb_event_t event,
                                                esp_ble_mesh_generic_server_cb_param_t *param);
```

Bluetooth Mesh Generic Server Model function.
Generic Server Model callback function type
- **Param event** Event type
- **Param param** Pointer to callback parameter

**Enumerations**

```c
enum esp_ble_mesh_generic_client_cb_event_t
```

This enum value is the event of Generic Client Model

*Values:*
- Enumerator `ESP_BLE_MESH_GENERIC_CLIENT_GET_STATE_EVT`
- Enumerator `ESP_BLE_MESH_GENERIC_CLIENT_SET_STATE_EVT`
- Enumerator `ESP_BLE_MESH_GENERIC_CLIENT_PUBLISH_EVT`
- Enumerator `ESP_BLE_MESH_GENERIC_CLIENT_TIMEOUT_EVT`
- Enumerator `ESP_BLE_MESH_GENERIC_CLIENT_EVT_MAX`

```c
enum esp_ble_mesh_gen_user_prop_access_t
```

This enum value is the access value of Generic User Property

*Values:*
- Enumerator `ESP_BLE_MESH_GEN_USER_ACCESS_PROHIBIT`
- Enumerator `ESP_BLE_MESH_GEN_USER_ACCESS_READ`
- Enumerator `ESP_BLE_MESH_GEN_USER_ACCESS_WRITE`
enumerator ESP_BLE_MESH_GEN_USER_ACCESS_READ_WRITE

enum esp_ble_mesh_gen_admin_prop_access_t
This enum value is the access value of Generic Admin Property
Values:
	enumerator ESP_BLE_MESH_GEN_ADMIN_NOT_USER_PROP
	enumerator ESP_BLE_MESH_GEN_ADMIN_ACCESS_READ
	enumerator ESP_BLE_MESH_GEN_ADMIN_ACCESS_WRITE
	enumerator ESP_BLE_MESH_GEN_ADMIN_ACCESS_READ_WRITE

enum esp_ble_mesh_gen_manu_prop_access_t
This enum value is the access value of Generic Manufacturer Property
Values:
	enumerator ESP_BLE_MESH_GEN_MANU_NOT_USER_PROP
	enumerator ESP_BLE_MESH_GEN_MANU_ACCESS_READ

tenumerato
Chapter 2. API 参考

Functions

```c
esp_err_t esp_ble_mesh_register_sensor_client_callback(esp_ble_mesh_sensor_client_cb_t callback)
```

Register BLE Mesh Sensor Client Model callback.

参数 callback - [in] Pointer to the callback function.

返回 ESP_OK on success or error code otherwise.

```c
esp_err_t esp_ble_mesh_sensor_client_get_state(esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_sensor_client_get_state_t *get_state)
```

Get the value of Sensor Server Model states using the Sensor Client Model get messages.

备注: If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_sensor_message_opcode_t in esp_ble_mesh_defs.h

```c
esp_err_t esp_ble_mesh_sensor_client_set_state(esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_sensor_client_set_state_t *set_state)
```

Set the value of Sensor Server Model states using the Sensor Client Model set messages.

备注: If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_sensor_message_opcode_t in esp_ble_mesh_defs.h

```c
union esp_ble_mesh_sensor_client_get_state_t
```

#include <esp_ble_mesh_sensor_model_api.h> Sensor Client Model get message union.

Public Members

```c
esp_ble_mesh_sensor_descriptor_get_t descriptor_get
```

For ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET

```c
esp_ble_mesh_sensor_cadence_get_t cadence_get
```

For ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET

Unions

union esp_ble_mesh_sensor_client_get_state_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Client Model get message union.
Chapter 2. API

**esp_ble_mesh_sensor_settings_get_t** settings_get
For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET

**esp_ble_mesh_sensor_setting_get_t** setting_get
For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_GET

**esp_ble_mesh_sensor_get_t** sensor_get
For ESP_BLE_MESH_MODEL_OP_SENSOR_GET

**esp_ble_mesh_sensor_column_get_t** column_get
For ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_GET

**esp_ble_mesh_sensor_series_get_t** series_get
For ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_GET

union **esp_ble_mesh_sensor_client_set_state_t**
#include <esp_ble_mesh_sensor_model_api.h> Sensor Client Model set message union.

**Public Members**

**esp_ble_mesh_sensor_cadence_set_t** cadence_set
For ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET & ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK

**esp_ble_mesh_sensor_setting_set_t** setting_set
For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET & ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET_UNACK

union **esp_ble_mesh_sensor_client_status_cb_t**
#include <esp_ble_mesh_sensor_model_api.h> Sensor Client Model received message union.

**Public Members**

**esp_ble_mesh_sensor_descriptor_status_cb_t** descriptor_status
For ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS

**esp_ble_mesh_sensor_cadence_status_cb_t** cadence_status
For ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_STATUS

**esp_ble_mesh_sensor_settings_status_cb_t** settings_status
For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_STATUS

**esp_ble_mesh_sensor_setting_status_cb_t** setting_status
For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_STATUS

**esp_ble_mesh_sensor_status_cb_t** sensor_status
For ESP_BLE_MESH_MODEL_OP_SENSOR_STATUS
esp_ble_mesh_sensor_column_status_cb_t column_status
For ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_STATUS

esp_ble_mesh_sensor_series_status_cb_t series_status
For ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_STATUS

union esp_ble_mesh_sensor_server_state_change_t
#include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model state change value union.

Public Members

esp_ble_mesh_state_change_sensor_cadence_set_t sensor_cadence_set
The recv_op in ctx can be used to decide which state is changed. Sensor Cadence Set

esp_ble_mesh_state_change_sensor_setting_set_t sensor_setting_set
Sensor Setting Set

union esp_ble_mesh_sensor_server_recv_get_msg_t
#include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model received get message union.

Public Members

esp_ble_mesh_server_recv_sensor_descriptor_get_t sensor_descriptor
Sensor Descriptor Get

esp_ble_mesh_server_recv_sensor_cadence_get_t sensor_cadence
Sensor Cadence Get

esp_ble_mesh_server_recv_sensor_settings_get_t sensor_settings
Sensor Settings Get

esp_ble_mesh_server_recv_sensor_setting_get_t sensor_setting
Sensor Setting Get

esp_ble_mesh_server_recv_sensor_get_t sensor_data
Sensor Get

esp_ble_mesh_server_recv_sensor_column_get_t sensor_column
Sensor Column Get

esp_ble_mesh_server_recv_sensor_series_get_t sensor_series
Sensor Series Get

union esp_ble_mesh_sensor_server_recv_set_msg_t
#include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model received set message union.
Chapter 2. API Reference

Public Members

```c
struct esp_ble_mesh_sensor_server_state_change_t
    state_change
ESPBLE_MESH_SENSOR_SERVER_STATE_CHANGE_EVT
```

```c
struct esp_ble_mesh_sensor_server_recv_get_msg_t
    get
ESPBLE_MESH_SENSOR_SERVER_RECV_GET_MSG_EVT
```

```c
struct esp_ble_mesh_sensor_server_recv_set_msg_t
    set
ESPBLE_MESH_SENSOR_SERVER_RECV_SET_MSG_EVT
```

Structures

```c
struct esp_ble_mesh_sensor_descriptor_get_t
    Bluetooth Mesh Sensor Client Model Get and Set parameters structure.
```

```c
struct esp_ble_mesh_sensor_cadence_get_t
    Parameter of Sensor Cadence Get
```

```c
struct esp_ble_mesh_sensor_cadence_set_t
    Parameters of Sensor Cadence Set
```

Public Members

```c
bool op_en
Indicate if optional parameters are included
```

```c
uint16_t property_id
Property ID of a sensor (optional)
```

```c
struct esp_ble_mesh_sensor_server_cb_value_t
    #include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model callback value union.
```

Public Members

```c
struct esp_ble_mesh_sensor_server_state_change_t
    state_change
ESP_BLE_MESH_SENSOR_SERVER_STATE_CHANGE_EVT
```

```c
struct esp_ble_mesh_sensor_server_recv_get_msg_t
    get
ESP_BLE_MESH_SENSOR_SERVER_RECV_GET_MSG_EVT
```

```c
struct esp_ble_mesh_sensor_server_recv_set_msg_t
    set
ESP_BLE_MESH_SENSOR_SERVER_RECV_SET_MSG_EVT
```
Chapter 2. API

Public Members

```c
uint16_t property_id
    Property ID for the sensor

uint8_t fast_cadence_period_divisor
    Divisor for the publish period

uint8_t status_trigger_type
    The unit and format of the Status Trigger Delta fields

struct net_buf_simple *status_trigger_delta_down
    Delta down value that triggers a status message

struct net_buf_simple *status_trigger_delta_up
    Delta up value that triggers a status message

uint8_t status_min_interval
    Minimum interval between two consecutive Status messages

struct net_buf_simple *fast_cadence_low
    Low value for the fast cadence range

struct net_buf_simple *fast_cadence_high
    Fast value for the fast cadence range
```

struct esp_ble_mesh_sensor_settings_get_t
    Parameter of Sensor Settings Get

Public Members

```c
uint16_t sensor_property_id
    Property ID of a sensor
```

struct esp_ble_mesh_sensor_setting_get_t
    Parameters of Sensor Setting Get

Public Members

```c
uint16_t sensor_property_id
    Property ID of a sensor

uint16_t sensor_setting_property_id
    Setting ID identifying a setting within a sensor
```

struct esp_ble_mesh_sensor_setting_set_t
    Parameters of Sensor Setting Set
Public Members

`uint16_t sensor_property_id`
Property ID identifying a sensor

`uint16_t sensor_setting_property_id`
Setting ID identifying a setting within a sensor

`struct net_buf_simple *sensor_setting_raw`
Raw value for the setting

`struct esp_ble_mesh_sensor_get_t`
Parameters of Sensor Get

Public Members

`bool op_en`
Indicate if optional parameters are included

`uint16_t property_id`
Property ID for the sensor (optional)

`struct esp_ble_mesh_sensor_column_get_t`
Parameters of Sensor Column Get

Public Members

`uint16_t property_id`
Property identifying a sensor

`struct net_buf_simple *raw_value_x`
Raw value identifying a column

`struct esp_ble_mesh_sensor_series_get_t`
Parameters of Sensor Series Get

Public Members

`bool op_en`
Indicate if optional parameters are included

`uint16_t property_id`
Property identifying a sensor

`struct net_buf_simple *raw_value_x1`
Raw value identifying a starting column (optional)
struct net_buf_simple *raw_value_x2
    Raw value identifying an ending column (C.1)

struct esp_ble_mesh_sensor_descriptor_status_cb_t
    Bluetooth Mesh Sensor Client Model Get and Set callback parameters structure.
    Parameter of Sensor Descriptor Status

    Public Members

    struct net_buf_simple *descriptor
        Sequence of 8-octet sensor descriptors (optional)

struct esp_ble_mesh_sensor_cadence_status_cb_t
    Parameters of Sensor Cadence Status

    Public Members

    uint16_t property_id
        Property for the sensor

    struct net_buf_simple *sensor_cadence_value
        Value of sensor cadence state

struct esp_ble_mesh_sensor_settings_status_cb_t
    Parameters of Sensor Settings Status

    Public Members

    uint16_t sensor_property_id
        Property ID identifying a sensor

    struct net_buf_simple *sensor_setting_property_ids
        A sequence of N sensor setting property IDs (optional)

struct esp_ble_mesh_sensor_setting_status_cb_t
    Parameters of Sensor Setting Status

    Public Members

    bool op_en
        Indicate id optional parameters are included

    uint16_t sensor_property_id
        Property ID identifying a sensor
Chapter 2. API

`uint16_t sensor_setting_property_id`
Setting ID identifying a setting within a sensor

`uint8_t sensor_setting_access`
Read/Write access rights for the setting (optional)

`struct net_buf_simple *sensor_setting_raw`
Raw value for the setting

`struct esp_ble_mesh_sensor_status_cb_t`
Parameter of Sensor Status

**Public Members**

`struct net_buf_simple *marshalled_sensor_data`
Value of sensor data state (optional)

`struct esp_ble_mesh_sensor_column_status_cb_t`
Parameters of Sensor Column Status

**Public Members**

`uint16_t property_id`
Property identifying a sensor and the Y axis

`struct net_buf_simple *sensor_column_value`
Left values of sensor column status

`struct esp_ble_mesh_sensor_series_status_cb_t`
Parameters of Sensor Series Status

**Public Members**

`uint16_t property_id`
Property identifying a sensor and the Y axis

`struct net_buf_simple *sensor_series_value`
Left values of sensor series status

`struct esp_ble_mesh_sensor_client_cb_param_t`
Sensor Client Model callback parameters

**Public Members**
Chapter 2. API

```c
int error_code

0: success, otherwise failure. For the error code values please refer to errno.h file. A negative sign is added to the standard error codes in errno.h.

```exp_ble_mesh_client_common_param_t *params```

The client common parameters.

```exp_ble_mesh_sensor_client_status_cb_t status_cb```

The sensor status message callback values

```struct esp_ble_mesh_sensor_descriptor_t```

Parameters of Sensor Descriptor state

### Public Members

```uint32_t positive_tolerance```

The value of Sensor Positive Tolerance field

```uint32_t negative_tolerance```

The value of Sensor Negative Tolerance field

```uint32_t sampling_function```

The value of Sensor Sampling Function field

```uint8_t measure_period```

The value of Sensor Measurement Period field

```uint8_t update_interval```

The value of Sensor Update Interval field

```struct esp_ble_mesh_sensor_setting_t```

Parameters of Sensor Setting state

### Public Members

```uint16_t property_id```

The value of Sensor Setting Property ID field

```uint8_t access```

The value of Sensor Setting Access field

```struct net_buf_simple *raw```

The value of Sensor Setting Raw field

```struct esp_ble_mesh_sensor_cadence_t```

Parameters of Sensor Cadence state
Public Members

```c
uint8_t period_divisor
    The value of Fast Cadence Period Divisor field
```

```c
uint8_t trigger_type
    The value of Status Trigger Type field
```

```c
struct net_buf_simple* trigger_delta_down
    Note: The parameter “size” in trigger_delta_down, trigger_delta_up, fast_cadence_low &
    fast_cadence_high indicates the exact length of these four parameters, and they are associated with the
    Sensor Property ID. Users need to initialize the “size” precisely. The value of Status Trigger Delta
    Down field
```

```c
struct net_buf_simple* trigger_delta_up
    The value of Status Trigger Delta Up field
```

```c
uint8_t min_interval
    The value of Status Min Interval field
```

```c
struct net_buf_simple* fast_cadence_low
    The value of Fast Cadence Low field
```

```c
struct net_buf_simple* fast_cadence_high
    The value of Fast Cadence High field
```

```c
struct esp_ble_mesh_sensor_data_t
    Parameters of Sensor Data state
```

Public Members

```c
uint8_t format
    Format A: The Length field is a 1-based uint4 value (valid range 0x0–0xF, representing range of 1 – 16).
    Format B: The Length field is a 1-based uint7 value (valid range 0x0–0x7F, representing range of 1 –
    127). The value 0x7F represents a length of zero. The value of the Sensor Data format
```

```c
uint8_t length
    The value of the Sensor Data length
```

```c
struct net_buf_simple* raw_value
    The value of Sensor Data raw value
```

```c
struct esp_ble_mesh_sensor_series_column_t
    Parameters of Sensor Series Column state
```

Public Members

Espressif Systems  Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
struct net_buf_simple *raw_value_x  
The value of Sensor Raw Value X field

struct net_buf_simple *column_width  
The value of Sensor Column Width field

struct net_buf_simple *raw_value_y  
The value of Sensor Raw Value Y field

struct esp_ble_mesh_sensor_state_t  
Parameters of Sensor states

Public Members

uint16_t sensor_property_id  
The value of Sensor Property ID field

esp_ble_mesh_sensor_descriptor_t descriptor  
Parameters of the Sensor Descriptor state

const uint8_t setting_count  
Multiple Sensor Setting states may be present for each sensor. The Sensor Setting Property ID values shall be unique for each Sensor Property ID that identifies a sensor within an element.

esp_ble_mesh_sensor_setting_t *settings  
Parameters of the Sensor Setting state

esp_ble_mesh_sensor_cadence_t *cadence  
The Sensor Cadence state may be not supported by sensors based on device properties referencing “non-scalar characteristics” such as “histograms” or “composite characteristics”. Parameters of the Sensor Cadence state

esp_ble_mesh_sensor_data_t sensor_data  
Parameters of the Sensor Data state

esp_ble_mesh_sensor_series_column_t series_column  
Parameters of the Sensor Series Column state

struct esp_ble_mesh_sensor_srv_t  
User data of Sensor Server Model

Public Members

esp_ble_mesh_model_t *model  
Pointer to the Sensor Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl  
Response control of the server model received messages
const uint8_t state_count
    Sensor state count

esp_ble_mesh_sensor_state_t *states
    Parameters of the Sensor states

struct esp_ble_mesh_sensor_setup_srv_t
    User data of Sensor Setup Server Model

Public Members

esp_ble_mesh_model_t *model
    Pointer to the Sensor Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl
    Response control of the server model received messages

const uint8_t state_count
    Sensor state count

esp_ble_mesh_sensor_state_t *states
    Parameters of the Sensor states

struct esp_ble_mesh_state_change_sensor_cadence_set_t
    Parameters of Sensor Cadence Set state change event

Public Members

uint16_t property_id
    The value of Sensor Property ID state

uint8_t period_divisor
    The value of Fast Cadence Period Divisor state

uint8_t trigger_type
    The value of Status Trigger Type state

struct net_buf_simple *trigger_delta_down
    The value of Status Trigger Delta Down state

struct net_buf_simple *trigger_delta_up
    The value of Status Trigger Delta Up state

uint8_t min_interval
    The value of Status Min Interval state
struct net_buf_simple *fast_cadence_low
The value of Fast Cadence Low state

struct net_buf_simple *fast_cadence_high
The value of Fast Cadence High state

struct esp_ble_mesh_state_change_sensor_setting_set_t
Parameters of Sensor Setting Set state change event

Public Members

uint16_t property_id
The value of Sensor Property ID state

uint16_t setting_property_id
The value of Sensor Setting Property ID state

struct net_buf_simple *setting_value
The value of Sensor Property Value state

struct esp_ble_mesh_server_recv_sensor_descriptor_get_t
Context of the received Sensor Descriptor Get message

Public Members

bool op_en
Indicate if optional parameters are included

uint16_t property_id
Property ID of a sensor (optional)

struct esp_ble_mesh_server_recv_sensor_cadence_get_t
Context of the received Sensor Cadence Get message

Public Members

uint16_t property_id
Property ID of a sensor

struct esp_ble_mesh_server_recv_sensor_settings_get_t
Context of the received Sensor Settings Get message

Public Members

uint16_t property_id
Property ID of a sensor
struct esp_ble_mesh_server_recv_sensor_setting_get_t
  Context of the received Sensor Setting Get message

  **Public Members**

  uint16_t property_id
  Property ID of a sensor

  uint16_t setting_property_id
  Setting ID identifying a setting within a sensor

struct esp_ble_mesh_server_recv_sensor_get_t
  Context of the received Sensor Get message

  **Public Members**

  bool op_en
  Indicate if optional parameters are included

  uint16_t property_id
  Property ID for the sensor (optional)

struct esp_ble_mesh_server_recv_sensor_column_get_t
  Context of the received Sensor Column Get message

  **Public Members**

  uint16_t property_id
  Property identifying a sensor

  struct net_buf_simple *raw_value_x
  Raw value identifying a column

struct esp_ble_mesh_server_recv_sensor_series_get_t
  Context of the received Sensor Series Get message

  **Public Members**

  bool op_en
  Indicate if optional parameters are included

  uint16_t property_id
  Property identifying a sensor

  struct net_buf_simple *raw_value
  Raw value containing X1 and X2 (optional)
struct esp_ble_mesh_server_recv_sensor_cadence_set_t
Context of the received Sensor Cadence Set message

Public Members

  uint16_t property_id
  Property ID for the sensor

  struct net_buf_simple *cadence
  Value of Sensor Cadence state

struct esp_ble_mesh_server_recv_sensor_setting_set_t
Context of the received Sensor Setting Set message

Public Members

  uint16_t property_id
  Property ID identifying a sensor

  uint16_t setting_property_id
  Setting ID identifying a setting within a sensor

  struct net_buf_simple *setting_raw
  Raw value for the setting

struct esp_ble_mesh_sensor_server_cb_param_t
Sensor Server Model callback parameters

Public Members

  esp_ble_mesh_model_t *model
  Pointer to Sensor Server Models

  esp_ble_mesh_msg_ctx_t ctx
  Context of the received messages

  esp_ble_mesh_sensor_server_cb_value_t value
  Value of the received Sensor Messages

Macros

ESP_BLE_MESH_MODEL_SENSOR_CLI (cli_pub, cli_data)
Define a new Sensor Client Model.

备注： This API needs to be called for each element on which the application needs to have a Sensor Client Model.
Chapter 2. API

参数
• cli_pub - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
• cli_data - Pointer to the unique struct `esp_ble_mesh_client_t`.

返回 New Sensor Client Model instance.

**ESP_BLE_MESH_MODEL_SENSOR_SRV** (srv_pub, srv_data)
Sensor Server Models related context.
Define a new Sensor Server Model.

备注: 1. The Sensor Server model is a root model. When this model is present on an element, the corresponding Sensor Setup Server model shall also be present.
   a. This model shall support model publication and model subscription.

参数
• srv_pub - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
• srv_data - Pointer to the unique struct `esp_ble_mesh_sensor srv_t`.

返回 New Sensor Server Model instance.

**ESP_BLE_MESH_MODEL_SENSOR_SETUP_SRV** (srv_pub, srv_data)
Define a new Sensor Setup Server Model.

   a. This model shall support model publication and model subscription.

参数
• srv_pub - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
• srv_data - Pointer to the unique struct `esp_ble_mesh_sensor_setup srv_t`.

返回 New Sensor Setup Server Model instance.

**ESP_BLE_MESH_INVALID_SENSOR_PROPERTY_ID**
Invalid Sensor Property ID

**ESP_BLE_MESH_SENSOR_PROPERTY_ID_LEN**
Length of Sensor Property ID

**ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN**
Length of Sensor Descriptor state

**ESP_BLE_MESH_SENSOR_UNSPECIFIED_POS_TOLERANCE**
Unspecified Sensor Positive Tolerance

**ESP_BLE_MESH_SENSOR_UNSPECIFIED_NEG_TOLERANCE**
Unspecified Sensor Negative Tolerance

**ESP_BLE_MESH_SENSOR_NOT_APPL_MEASURE_PERIOD**
Not applicable Sensor Measurement Period

**ESP_BLE_MESH_SENSOR_NOT_APPL_UPDATE_INTERVAL**
Not applicable Sensor Update Interval
ESP_BLE_MESH_INVALID_SENSOR_SETTING_PROPERTY_ID
Invalid Sensor Setting Property ID

ESP_BLE_MESH_SENSOR_SETTING_PROPERTY_ID_LEN
Length of Sensor Setting Property ID

ESP_BLE_MESH_SENSOR_SETTING_ACCESS_LEN
Length of Sensor Setting Access

ESP_BLE_MESH_SENSOR_SETTING_ACCESS_READ
Sensor Setting Access - Read

ESP_BLE_MESH_SENSOR_SETTING_ACCESS_READ_WRITE
Sensor Setting Access - Read & Write

ESP_BLE_MESH_SENSOR_DIVISOR_TRIGGER_TYPE_LEN
Length of Sensor Divisor Trigger Type

ESP_BLE_MESH_SENSOR_STATUS_MIN_INTERVAL_LEN
Length of Sensor Status Min Interval

ESP_BLE_MESH_SENSOR_PERIOD_DIVISOR_MAX_VALUE
Maximum value of Sensor Period Divisor

ESP_BLE_MESH_SENSOR_STATUS_MIN_INTERVAL_MAX
Maximum value of Sensor Status Min Interval

ESP_BLE_MESH_SENSOR_STATUS_TRIGGER_TYPE_CHAR
Sensor Status Trigger Type - Format Type of the characteristic that the Sensor Property ID state references

ESP_BLE_MESH_SENSOR_STATUS_TRIGGER_TYPE_UINT16
Sensor Status Trigger Type - Format Type “uint16”

ESP_BLE_MESH_SENSOR_DATA_FORMAT_A
Sensor Data Format A

ESP_BLE_MESH_SENSOR_DATA_FORMAT_B
Sensor Data Format B

ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID_LEN
MPID length of Sensor Data Format A

ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN
MPID length of Sensor Data Format B

ESP_BLE_MESH_SENSOR_DATA_ZERO_LEN
Zero length of Sensor Data.
Note: The Length field is a 1-based uint7 value (valid range 0x0-0x7F, representing range of 1–127). The value 0x7F represents a length of zero.
**ESP_BLE_MESH_GETSENSOR_DATA_FORMAT** (_data_

Get format of the sensor data.

**备注:** Multiple sensor data may be concatenated. Make sure the _data pointer is updated before getting the format of the corresponding sensor data.

**参数**

- _data_ – Pointer to the start of the sensor data.

**返回** Format of the sensor data.

**ESP_BLE_MESH_GETSENSOR_DATA_LENGTH** (_data_, _fmt_

Get length of the sensor data.

**备注:** Multiple sensor data may be concatenated. Make sure the _data pointer is updated before getting the length of the corresponding sensor data.

**参数**

- _data_ – Pointer to the start of the sensor data.
- _fmt_ – Format of the sensor data.

**返回** Length (zero-based) of the sensor data.

**ESP_BLE_MESH_GETSENSOR_DATA_PROPERTY_ID** (_data_, _fmt_

Get Sensor Property ID of the sensor data.

**备注:** Multiple sensor data may be concatenated. Make sure the _data pointer is updated before getting Sensor Property ID of the corresponding sensor data.

**参数**

- _data_ – Pointer to the start of the sensor data.
- _fmt_ – Format of the sensor data.

**返回** Sensor Property ID of the sensor data.

**ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID** (_len_, _id_

Generate a MPID value for sensor data with Format A.

**备注:**

1. The Format field is 0b0 and indicates that Format A is used.
   a. The Length field is a 1-based uint4 value (valid range 0x0–0xF, representing range of 1–16).
   b. The Property ID is an 11-bit bit field representing 11 LSb of a Property ID.
   c. This format may be used for Property Values that are not longer than 16 octets and for Property IDs less than 0x800.

**参数**

- _len_ – Length of Sensor Raw value.
- _id_ – Sensor Property ID.

**返回** 2-octet MPID value for sensor data with Format A.

**ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID** (_len_, _id_

Generate a MPID value for sensor data with Format B.

**备注:**

1. The Format field is 0b1 and indicates Format B is used.
a. The Length field is a 1-based uint7 value (valid range 0x0-0x7F, representing range of 1–127). The value 0x7F represents a length of zero.
b. The Property ID is a 16-bit bit field representing a Property ID.
c. This format may be used for Property Values not longer than 128 octets and for any Property IDs. Property values longer than 128 octets are not supported by the Sensor Status message.
d. Exclude the generated 1-octet value, the 2-octet Sensor Property ID.

参数
- \_len - Length of Sensor Raw value.
- \_id - Sensor Property ID.

返回 3-octet MPID value for sensor data with Format B.

Type Definitions

typedef void (*esp_ble_mesh_sensor_client_cb_t)(esp_ble_mesh_sensor_client_cb_event_t event, esp_ble_mesh_sensor_client_cb_param_t *param)

Bluetooth Mesh Sensor Client Model function.

Sensor Client Model callback function type

Param event Event type
Param param Pointer to callback parameter

typedef void (*esp_ble_mesh_sensor_server_cb_t)(esp_ble_mesh_sensor_server_cb_event_t event, esp_ble_mesh_sensor_server_cb_param_t *param)

Bluetooth Mesh Sensor Server Model function.

Sensor Server Model callback function type

Param event Event type
Param param Pointer to callback parameter

Enumerations
enum esp_ble_mesh_sensor_client_cb_event_t

This enum value is the event of Sensor Client Model

Values:

enumerator ESP_BLE_MESH_SENSOR_CLIENT_GET_STATE_EVT
enumerator ESP_BLE_MESH_SENSOR_CLIENT_SET_STATE_EVT
enumerator ESP_BLE_MESH_SENSOR_CLIENT_PUBLISH_EVT
enumerator ESP_BLE_MESH_SENSOR_CLIENT_TIMEOUT_EVT
enumerator ESP_BLE_MESH_SENSOR_CLIENT_EVT_MAX

denum esp_ble_mesh_sensor_sample_func

This enum value is value of Sensor Sampling Function

Values:

enumerator ESP_BLE_MESH_SAMPLE_FUNC_UNSPECIFIED
enumerator ESP_BLE_MESH_SAMPLE_FUNC_INSTANTANEOUS
enumerator ESP_BLE_MESH_SAMPLE_FUNC_ARITHMETIC_MEAN
enumerator ESP_BLE_MESH_SAMPLE_FUNC_RMS
enumerator ESP_BLE_MESH_SAMPLE_FUNC_MAXIMUM
enumerator ESP_BLE_MESH_SAMPLE_FUNC_MINIMUM
enumerator ESP_BLE_MESH_SAMPLE_FUNC_ACCUMULATED
enumerator ESP_BLE_MESH_SAMPLE_FUNC_COUNT

enum esp_ble_mesh_sensor_server_cb_event_t

This enum value is the event of Sensor Server Model

Values:

enumerator ESP_BLE_MESH_SENSOR_SERVER_STATE_CHANGE_EVT

i. When get_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, no event will be callback to the application layer when Sensor Get messages are received.
ii. When set_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, this event will be callback to the application layer when Sensor Set/Set Unack messages are received.

enumerator ESP_BLE_MESH_SENSOR_SERVER_RECV_GET_MSG_EVT

When get_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Sensor Get messages are received.

enumerator ESP_BLE_MESH_SENSOR_SERVER_RECV_SET_MSG_EVT

When set_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Sensor Set/Set Unack messages are received.

enumerator ESP_BLE_MESH_SENSOR_SERVER_EVT_MAX

Time and Scenes Client/Server Models

Header File

- components/bt/esp_ble_mesh/api/models/include/esp_ble_mesh_time_scene_model_api.h

Functions

esp_err_t esp_ble_mesh_register_time_scene_client_callback(esp_ble_mesh_time_scene_client_cb_t callback)

Register BLE Mesh Time Scene Client Model callback.

参数 callback [in] Pointer to the callback function.
返回 ESP_OK on success or error code otherwise.
\texttt{esp_err_t esp_ble_mesh_time_scene_client_get_state(esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_time_scene_client_get_state_t *get\_state)}

Get the value of Time Scene Server Model states using the Time Scene Client Model get messages.

**备注:** If you want to know the opcodes and corresponding meanings accepted by this API, please refer to \texttt{esp\_ble\_mesh\_time\_scene\_message\_opcode_t} in \texttt{esp\_ble\_mesh\_defs.h}

参数

- **params** - [in] Pointer to BLE Mesh common client parameters.
- **get\_state** - [in] Pointer to time scene get message value. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

\texttt{esp_err_t esp_ble_mesh_time_scene_client_set_state(esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_time_scene_client_set_state_t *set\_state)}

Set the value of Time Scene Server Model states using the Time Scene Client Model set messages.

**备注:** If you want to know the opcodes and corresponding meanings accepted by this API, please refer to \texttt{esp\_ble\_mesh\_time\_scene\_message\_opcode_t} in \texttt{esp\_ble\_mesh\_defs.h}

参数

- **params** - [in] Pointer to BLE Mesh common client parameters.
- **set\_state** - [in] Pointer to time scene set message value. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

\texttt{esp_err_t esp_ble_mesh_register_time_scene_server_callback(esp_ble_mesh_time_scene_server_cb_t callback)}

Register BLE Mesh Time and Scenes Server Model callback.

参数 **callback** - [in] Pointer to the callback function.

返回 ESP_OK on success or error code otherwise.

Unions

\texttt{union \texttt{esp\_ble\_mesh\_time\_scene\_client\_get\_state\_t} #include <esp\_ble\_mesh\_time\_scene\_model\_api.h> Time Scene Client Model get message union.}

Public Members

\texttt{\texttt{esp\_ble\_mesh\_scheduler\_act\_get} scheduler\_act\_get}

For ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_GET

\texttt{union \texttt{esp\_ble\_mesh\_time\_scene\_client\_set\_state\_t} #include <esp\_ble\_mesh\_time\_scene\_model\_api.h> Time Scene Client Model set message union.}

Public Members
Esp BLE mesh time set_t time_set
For ESP BLE_MESH_MODEL_OP_TIME_SET

Esp BLE mesh time zone set_t time_zone_set
For ESP BLE_MESH_MODEL_OP_TIME_ZONE_SET

Esp BLE mesh tai utc delta set_t tai_utc_delta_set
For ESP BLE_MESH_MODEL_OP_TAI.UTC_DELTA_SET

Esp BLE mesh time role set_t time_role_set
For ESP BLE_MESH_MODEL_OP_TIME_ROLE_SET

Esp BLE mesh scene store_t scene_store
For ESP_BLE_MESH_MODEL_OP_SCENE_STORE & ESP_BLE_MESH_MODEL_OP_SCENE_STORE_UNACK

Esp BLE mesh scene recall_t scene_recall
For ESP_BLE_MESH_MODEL_OP_SCENE_RECALL & ESP_BLE_MESH_MODEL_OP_SCENE_RECALL_UNACK

Esp BLE mesh scene delete_t scene_delete
For ESP_BLE_MESH_MODEL_OP_SCENE_DELETE & ESP_BLE_MESH_MODEL_OP_SCENE_DELETE_UNACK

Esp BLE mesh scheduler act set_t scheduler_act_set
For ESP_BLE_MESH_MODEL_OP_SCHEDULER.ACT_SET &
ESP_BLE_MESH_MODEL_OP_SCHEDULER.ACT_SET_UNACK

Union esp BLE mesh time scene client status cb_t
#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Client Model received message union.

Public Members

Esp BLE mesh time status cb_t time_status
For ESP_BLE_MESH_MODEL_OP_TIME_STATUS

Esp BLE mesh time zone status cb_t time_zone_status
For ESP_BLE_MESH_MODEL_OP_TIME_ZONE_STATUS

Esp BLE mesh tai utc delta status cb_t tai_utc_delta_status
For ESP_BLE_MESH_MODEL_OP_TAI.UTC_DELTA_STATUS

Esp BLE mesh time role status cb_t time_role_status
For ESP_BLE_MESH_MODEL_OP_TIME_ROLE_STATUS

Esp BLE mesh scene status cb_t scene_status
For ESP_BLE_MESH_MODEL_OP_SCENE_STATUS

Esp BLE mesh scene register status cb_t scene_register_status
For ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_STATUS
Chapter 2. API 参考

```c
esp_ble_mesh_scheduler_status_cb_t scheduler_status
For ESP_BLE_MESH_MODEL_OP_SCHEDULER_STATUS
```

```c
esp_ble_mesh_scheduler_act_status_cb_t scheduler_act_status
For ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_STATUS
```

union esp_ble_mesh_time_scene_server_state_change_t
#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model state change value union.

### Public Members

```c
esp_ble_mesh_state_change_time_set_t time_set
The recv_op in ctx can be used to decide which state is changed. Time Set
```

```c
esp_ble_mesh_state_change_time_status_t time_status
Time Status
```

```c
esp_ble_mesh_state_change_time_zone_set_t time_zone_set
Time Zone Set
```

```c
esp_ble_mesh_state_change_tai_utc_delta_set_t tai_utc_delta_set
TAI UTC Delta Set
```

```c
esp_ble_mesh_state_change_time_role_set_t time_role_set
Time Role Set
```

```c
esp_ble_mesh_state_change_scene_store_t scene_store
Scene Store
```

```c
esp_ble_mesh_state_change_scene_recall_t scene_recall
Scene Recall
```

```c
esp_ble_mesh_state_change_scene_delete_t scene_delete
Scene Delete
```

```c
esp_ble_mesh_state_change_scheduler_act_set_t scheduler_act_set
Scheduler Action Set
```

union esp_ble_mesh_time_scene_server_recv_get_msg_t
#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model received get message union.

### Public Members

```c
esp_ble_mesh_server_recv_scheduler_act_get_t scheduler_act
Scheduler Action Get
```

union esp_ble_mesh_time_scene_server_recv_set_msg_t
#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model received set message union.
Public Members

`esp_ble_mesh_server_recv_time_set_t` **time**
Time Set

`esp_ble_mesh_server_recv_time_zone_set_t` **time_zone**
Time Zone Set

`esp_ble_mesh_server_recv_tai_utc_delta_set_t` **tai_utc_delta**
TAI-UTC Delta Set

`esp_ble_mesh_server_recv_time_role_set_t` **time_role**
Time Role Set

`esp_ble_mesh_server_recv_scene_store_t` **scene_store**
Scene Store/Scene Store Unack

`esp_ble_mesh_server_recv_scene_recall_t` **scene_recall**
Scene Recall/Scene Recall Unack

`esp_ble_mesh_server_recv_scene_delete_t` **scene_delete**
Scene Delete/Scene Delete Unack

`esp_ble_mesh_server_recv_scheduler_act_set_t` **scheduler_act**
Scheduler Action Set/Scheduler Action Set Unack

union `esp_ble_mesh_time_scene_server_recv_status_msg_t`
#include `<esp_ble_mesh_time_scene_model_api.h>` Time Scene Server Model received status message union.

Public Members

`esp_ble_mesh_server_recv_time_status_t` **time_status**
Time Status

union `esp_ble_mesh_time_scene_server_cb_value_t`
#include `<esp_ble_mesh_time_scene_model_api.h>` Time Scene Server Model callback value union.

Public Members

`esp_ble_mesh_time_scene_server_state_change_t` **state_change**
ESP_BLE_MESH_TIME_SCENE_SERVER_STATE_CHANGE_EVT

`esp_ble_mesh_time_scene_server_recv_get_msg_t` **get**
ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_GET_MSG_EVT

`esp_ble_mesh_time_scene_server_recv_set_msg_t` **set**
ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_SET_MSG_EVT
\textit{esp_ble_mesh_time_scene_server_recv_status_msg_t} \textit{status} ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_STATUS_MSG_EVT

\textbf{Structures}

\textbf{struct \textit{esp_ble_mesh_time_set_t}}

Bluetooth Mesh Time Scene Client Model Get and Set parameters structure.

Parameters of Time Set

\textbf{Public Members}

\begin{verbatim}
    uint8_t \textit{tai_seconds}[5]
    \textit{The current TAI time in seconds}

    uint8_t \textit{sub_second}
    \textit{The sub-second time in units of 1/256 second}

    uint8_t \textit{uncertainty}
    \textit{The estimated uncertainty in 10-millisecond steps}

    uint16_t \textit{time_authority}
    \textit{0 = No Time Authority, 1 = Time Authority}

    uint16_t \textit{tai_utc_delta}
    \textit{Current difference between TAI and UTC in seconds}

    uint8_t \textit{time_zone_offset}
    \textit{The local timezone offset in 15-minute increments}
\end{verbatim}

struct \textit{esp_ble_mesh_time_zone_set_t}

Parameters of Time Zone Set

\textbf{Public Members}

\begin{verbatim}
    uint8_t \textit{time_zone_offset_new}
    \textit{Upcoming local timezone offset}

    uint8_t \textit{tai_zone_change}[5]
    \textit{TAI Second time of the upcoming Time Zone Offset change}
\end{verbatim}

struct \textit{esp_ble_mesh_tai_utc_delta_set_t}

Parameters of TAI-UTC Delta Set

\textbf{Public Members}

\begin{verbatim}
    uint16_t \textit{tai_utc_delta_new}
    \textit{Upcoming difference between TAI and UTC in seconds}
\end{verbatim}
Chapter 2. API

uint16_t padding
    Always 0b0. Other values are Prohibited.

uint8_t tai_delta_change[5]
    TAI Second time of the upcoming TAI-UTC Delta change

struct esp_ble_mesh_time_role_set_t
    Parameter of Time Role Set

Public Members

uint8_t time_role
    The Time Role for the element

struct esp_ble_mesh_scene_store_t
    Parameter of Scene Store

Public Members

uint16_t scene_number
    The number of scenes to be stored

struct esp_ble_mesh_scene_recall_t
    Parameters of Scene Recall

Public Members

bool op_en
    Indicate if optional parameters are included

uint16_t scene_number
    The number of scenes to be recalled

uint8_t tid
    Transaction ID

uint8_t trans_time
    Time to complete state transition (optional)

uint8_t delay
    Indicate message execution delay (C.1)

struct esp_ble_mesh_scene_delete_t
    Parameter of Scene Delete
Chapter 2. API 参考

**Public Members**

```c
uint16_t scene_number
```

The number of scenes to be deleted

**struct esp_ble_mesh_scheduler_act_get_t**

Parameter of Scheduler Action Get

**Public Members**

```c
uint8_t index
```

Index of the Schedule Register entry to get

**struct esp_ble_mesh_scheduler_act_set_t**

Parameters of Scheduler Action Set

**Public Members**

```c
uint64_t index
```

Index of the Schedule Register entry to set

```c
uint64_t year
```

Scheduled year for the action

```c
uint64_t month
```

Scheduled month for the action

```c
uint64_t day
```

Scheduled day of the month for the action

```c
uint64_t hour
```

Scheduled hour for the action

```c
uint64_t minute
```

Scheduled minute for the action

```c
uint64_t second
```

Scheduled second for the action

```c
uint64_t day_of_week
```

Schedule days of the week for the action

```c
uint64_t action
```

Action to be performed at the scheduled time

```c
uint64_t trans_time
```

Transition time for this action
uint16_t scene_number
    Transition time for this action

struct esp_ble_mesh_time_status_cb_t
    Bluetooth Mesh Time Scene Client Model Get and Set callback parameters structure.

Parameters of Time Status

Public Members

uint8_t tai_seconds[5]
    The current TAI time in seconds

uint8_t sub_second
    The sub-second time in units of 1/256 second

uint8_t uncertainty
    The estimated uncertainty in 10-millisecond steps

uint16_t time_authority
    0 = No Time Authority, 1 = Time Authority

uint16_t tai_utc_delta
    Current difference between TAI and UTC in seconds

uint8_t time_zone_offset
    The local time zone offset in 15-minute increments

struct esp_ble_mesh_time_zone_status_cb_t
    Parameters of Time Zone Status

Public Members

uint8_t time_zone_offset_curr
    Current local time zone offset

uint8_t time_zone_offset_new
    Upcoming local time zone offset

uint8_t tai_zone_change[5]
    TAI Second time of the upcoming Time Zone Offset change

struct esp_ble_mesh_tai_utc_delta_status_cb_t
    Parameters of TAI-UTC Delta Status
Public Members

uint16_t **tai_utc_delta_curr**
Current difference between TAI and UTC in seconds

uint16_t **padding_1**
Always 0b0. Other values are Prohibited.

uint16_t **tai_utc_delta_new**
Upcoming difference between TAI and UTC in seconds

uint16_t **padding_2**
Always 0b0. Other values are Prohibited.

uint8_t **tai_delta_change[5]**
TAI Seconds time of the upcoming TAI-UTC Delta change

struct **esp_ble_mesh_time_role_status_cb_t**
Parameter of Time Role Status

Public Members

uint8_t **time_role**
The Time Role for the element

struct **esp_ble_mesh_scene_status_cb_t**
Parameters of Scene Status

Public Members

bool **op_en**
Indicate if optional parameters are included

uint8_t **status_code**
Status code of the last operation

uint16_t **current_scene**
Scene Number of the current scene

uint16_t **target_scene**
Scene Number of the target scene (optional)

uint8_t **remain_time**
Time to complete state transition (C.1)

struct **esp_ble_mesh_scene_register_status_cb_t**
Parameters of Scene Register Status
Public Members

uint8_t **status_code
Status code for the previous operation

tuint16_t **current_scene
Scene Number of the current scene

struct net_buf_simple **scenes
A list of scenes stored within an element

struct esp_ble_mesh_scheduler_status_cb_t
Parameter of Scheduler Status

Public Members

uint16_t **schedules
Bit field indicating defined Actions in the Schedule Register

struct esp_ble_mesh_scheduler_act_status_cb_t
Parameters of Scheduler Action Status

Public Members

uint64_t **index
Enumerates (selects) a Schedule Register entry

uint64_t **year
Scheduled year for the action

uint64_t **month
Scheduled month for the action

uint64_t **day
Scheduled day of the month for the action

uint64_t **hour
Scheduled hour for the action

uint64_t **minute
Scheduled minute for the action

uint64_t **second
Scheduled second for the action

uint64_t **day_of_week
Scheduled days of the week for the action
uint64_t action
    Action to be performed at the scheduled time

uint64_t trans_time
    Transition time for this action

uint16_t scene_number
    Transition time for this action

struct esp_ble_mesh_time_scene_client_cb_param_t
    Time Scene Client Model callback parameters

Public Members

int error_code
    Appropriate error code

esp_ble_mesh_client_common_param_t *params
    The client common parameters.

esp_ble_mesh_time_scene_client_status_cb_t status_cb
    The scene status message callback values

struct esp_ble_mesh_time_state_t
    Parameters of Time state

Public Members

uint8_t tai_seconds[5]
    The value of the TAI Seconds state

uint8_t subsecond
    The value of the Subsecond field

uint8_t uncertainty
    The value of the Uncertainty field

uint8_t timezone_offset_curr
    The value of the Time Zone Offset Current field

uint8_t timezone_offset_new
    The value of the Time Zone Offset New state

uint8_t tai_zone_change[5]
    The value of the TAI of Zone Chaneg field
uint16_t time_authority
    The value of the Time Authority bit

uint16_t tai_utc_delta_curr
    The value of the TAI-UTC Delta Current state

uint16_t tai_utc_delta_new
    The value of the TAI-UTC Delta New state

uint8_t tai_delta_change[5]
    The value of the TAI of Delta Change field

struct esp_ble_mesh_time_state_t::[anonymous] time
    Parameters of the Time state

uint8_t time_role
    The value of the Time Role state

struct esp_ble_mesh_time_srv_t
    User data of Time Server Model

Public Members

esp_ble_mesh_model_t *model
    Pointer to the Time Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl
    Response control of the server model received messages

esp_ble_mesh_time_state_t *state
    Parameters of the Time state

struct esp_ble_mesh_time_setup_srv_t
    User data of Time Setup Server Model

Public Members

esp_ble_mesh_model_t *model
    Pointer to the Time Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl
    Response control of the server model received messages

esp_ble_mesh_time_state_t *state
    Parameters of the Time state
struct `esp_ble_mesh_scene_register_t`

a. Scene Store is an operation of storing values of a present state of an element.
b. The structure and meaning of the stored state is determined by a model. States to be stored are specified by each model.
c. The Scene Store operation shall persistently store all values of all states marked as Stored with Scene for all models present on all elements of a node.
d. If a model is extending another model, the extending model shall determine the Stored with Scene behavior of that model. Parameters of Scene Register state.

**Public Members**

`uint16_t scene_number`
- The value of the Scene Number.

`uint8_t scene_type`
- The value of the Scene Type.

`struct net_buf_simple *scene_value`
- Scene value may use a union to represent later, the union contains structures of all the model states which can be stored in a scene. The value of the Scene Value.

struct `esp_ble_mesh_scenes_state_t`

Parameters of Scenes state.

Scenes serve as memory banks for storage of states (e.g., a power level or a light level/color). Values of states of an element can be stored as a scene and can be recalled later from the scene memory.

A scene is represented by a Scene Number, which is a 16-bit non-zero, mesh-wide value. (There can be a maximum of 65535 scenes in a mesh network.) The meaning of a scene, as well as the state storage container associated with it, are determined by a model.

The Scenes state change may start numerous parallel model transitions. In that case, each individual model handles the transition internally.

The scene transition is defined as a group of individual model transitions started by a Scene Recall operation. The scene transition is in progress when at least one transition from the group of individual model transitions is in progress.

**Public Members**

`const uint16_t scene_count`
- The Scenes state’s scene count.

`esp_ble_mesh_scene_register_t *scenes`
- Parameters of the Scenes state.

`uint16_t current_scene`
- The Current Scene state is a 16-bit value that contains either the Scene Number of the currently active scene or a value of 0x0000 when no scene is active.

When a Scene Store operation or a Scene Recall operation completes with success, the Current Scene state value shall be to the Scene Number used during that operation.
When the Current Scene Number is deleted from a Scene Register state as a result of Scene Delete operation, the Current Scene state shall be set to 0x0000.

When any of the element’s state that is marked as “Stored with Scene” has changed not as a result of a Scene Recall operation, the value of the Current Scene state shall be set to 0x0000.

When a scene transition is in progress, the value of the Current Scene state shall be set to 0x0000. The value of the Current Scene state

\texttt{uint16_t target\_scene}

The Target Scene state is a 16-bit value that contains the target Scene Number when a scene transition is in progress.

When the scene transition is in progress and the target Scene Number is deleted from a Scene Register state as a result of Scene Delete operation, the Target Scene state shall be set to 0x0000.

When the scene transition is in progress and a new Scene Number is stored in the Scene Register as a result of Scene Store operation, the Target Scene state shall be set to the new Scene Number.

When the scene transition is not in progress, the value of the Target Scene state shall be set to 0x0000. The value of the Target Scene state

\texttt{uint8_t status\_code}

The status code of the last scene operation

\texttt{bool in\_progress}

Indicate if the scene transition is in progress

\texttt{struct esp\_ble\_mesh\_scene\_srv\_t}

User data of Scene Server Model

**Public Members**

\texttt{esp\_ble\_mesh\_model\_t *model}

Pointer to the Scene Server Model. Initialized internally.

\texttt{esp\_ble\_mesh\_server\_rsp\_ctrl\_t *rsp\_ctrl}

Response control of the server model received messages

\texttt{esp\_ble\_mesh\_scenes\_state\_t *state}

Parameters of the Scenes state

\texttt{esp\_ble\_mesh\_last\_msg\_info\_t *last}

Parameters of the last received set message

\texttt{esp\_ble\_mesh\_state\_transition\_t *transition}

Parameters of state transition

\texttt{struct esp\_ble\_mesh\_scene\_setup\_srv\_t}

User data of Scene Setup Server Model
Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Scene Setup Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`esp_ble_mesh_scenes_state_t *state`
Parameters of the Scenes state

`struct esp_ble_mesh_schedule_register_t`
Parameters of Scheduler Register state

Public Members

`bool in_use`
Indicate if the registered schedule is in use

`uint64_t year`
The value of Scheduled year for the action

`uint64_t month`
The value of Scheduled month for the action

`uint64_t day`
The value of Scheduled day of the month for the action

`uint64_t hour`
The value of Scheduled hour for the action

`uint64_t minute`
The value of Scheduled minute for the action

`uint64_t second`
The value of Scheduled second for the action

`uint64_t day_of_week`
The value of Schedule days of the week for the action

`uint64_t action`
The value of Action to be performed at the scheduled time

`uint64_t trans_time`
The value of Transition time for this action

`uint16_t scene_number`
The value of Scene Number to be used for some actions
struct `esp_ble_mesh_scheduler_state_t`  
Parameters of Scheduler state

**Public Members**

```c
const uint8_t schedule_count
```
Scheduler count

```c
esp_ble_mesh_schedule_register_t *schedules
```
Up to 16 scheduled entries

struct `esp_ble_mesh_scheduler_srv_t`  
User data of Scheduler Server Model

**Public Members**

```c
esp_ble_mesh_model_t *model
```
Pointer to the Scheduler Server Model. Initialized internally.

```c
esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl
```
Response control of the server model received messages

```c
esp_ble_mesh_scheduler_state_t *state
```
Parameters of the Scheduler state

struct `esp_ble_mesh_scheduler_setup_srv_t`  
User data of Scheduler Setup Server Model

**Public Members**

```c
esp_ble_mesh_model_t *model
```
Pointer to the Scheduler Setup Server Model. Initialized internally.

```c
esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl
```
Response control of the server model received messages

```c
esp_ble_mesh_scheduler_state_t *state
```
Parameters of the Scheduler state

struct `esp_ble_mesh_state_change_time_set_t`  
Parameters of Time Set state change event

**Public Members**

```c
uint8_t tai_seconds[5]
```
The current TAI time in seconds
uint8_t _subsecond
The sub-second time in units of 1/256 second

uint8_t _uncertainty
The estimated uncertainty in 10-millisecond steps

uint16_t _time_authority
0 = No Time Authority, 1 = Time Authority

uint16_t _tai_utc_delta_curr
Current difference between TAI and UTC in seconds

uint8_t _time_zone_offset_curr
The local time zone offset in 15-minute increments

struct esp_ble_mesh_state_change_time_status_t
Parameters of Time Status state change event

Public Members

uint8_t _tai_seconds[5]
The current TAI time in seconds

uint8_t _subsecond
The sub-second time in units of 1/256 second

uint8_t _uncertainty
The estimated uncertainty in 10-millisecond steps

uint16_t _time_authority
0 = No Time Authority, 1 = Time Authority

uint16_t _tai_utc_delta_curr
Current difference between TAI and UTC in seconds

uint8_t _time_zone_offset_curr
The local time zone offset in 15-minute increments

struct esp_ble_mesh_state_change_time_zone_set_t
Parameters of Time Zone Set state change event

Public Members

uint8_t _time_zone_offset_new
Upcoming local time zone offset
uint8_t \texttt{tai\_zone\_change}[5]
TAI Seconds time of the upcoming Time Zone Offset change

struct \texttt{esp\_ble\_mesh\_state\_change\_tai\_utc\_delta\_set\_t}
Parameters of TAI UTC Delta Set state change event

**Public Members**

uint16_t \texttt{tai\_utc\_delta\_new}
Upcoming difference between TAI and UTC in seconds

uint8_t \texttt{tai\_delta\_change}[5]
TAI Seconds time of the upcoming TAI-UTC Delta change

struct \texttt{esp\_ble\_mesh\_state\_change\_time\_role\_set\_t}
Parameter of Time Role Set state change event

**Public Members**

uint8_t \texttt{time\_role}
The Time Role for the element

struct \texttt{esp\_ble\_mesh\_state\_change\_scene\_store\_t}
Parameter of Scene Store state change event

**Public Members**

uint16_t \texttt{scene\_number}
The number of scenes to be stored

struct \texttt{esp\_ble\_mesh\_state\_change\_scene\_recall\_t}
Parameter of Scene Recall state change event

**Public Members**

uint16_t \texttt{scene\_number}
The number of scenes to be recalled

struct \texttt{esp\_ble\_mesh\_state\_change\_scene\_delete\_t}
Parameter of Scene Delete state change event

**Public Members**

uint16_t \texttt{scene\_number}
The number of scenes to be deleted
struct esp_ble_mesh_state_change_scheduler_act_set_t
    Parameter of Scheduler Action Set state change event

Public Members

uint64_t index
    Index of the Schedule Register entry to set

uint64_t year
    Scheduled year for the action

uint64_t month
    Scheduled month for the action

uint64_t day
    Scheduled day of the month for the action

uint64_t hour
    Scheduled hour for the action

uint64_t minute
    Scheduled minute for the action

uint64_t second
    Scheduled second for the action

uint64_t day_of_week
    Schedule days of the week for the action

uint64_t action
    Action to be performed at the scheduled time

uint64_t trans_time
    Transition time for this action

uint16_t scene_number
    Scene number to be used for some actions

struct esp_ble_mesh_server_recv_scheduler_act_get_t
    Context of the received Scheduler Action Get message

Public Members

uint8_t index
    Index of the Schedule Register entry to get

struct esp_ble_mesh_server_recv_time_set_t
    Context of the received Time Set message
Public Members

uint8_t **tai_seconds**[5]
   The current TAI time in seconds

uint8_t **subsecond**
   The sub-second time in units of 1/256 second

uint8_t **uncertainty**
   The estimated uncertainty in 10-millisecond steps

uint16_t **time_authority**
   0 = No Time Authority, 1 = Time Authority

uint16_t **tai_utc_delta**
   Current difference between TAI and UTC in seconds

uint8_t **time_zone_offset**
   The local time zone offset in 15-minute increments

struct esp_ble_mesh_server_recv_time_zone_set_t
   Context of the received TimeZone Set message

Public Members

uint8_t **time_zone_offset_new**
   Upcoming local time zone offset

uint8_t **tai_zone_change**[5]
   TAI Seconds time of the upcoming Time Zone Offset change

struct esp_ble_mesh_server_recv_tai_utc_delta_set_t
   Context of the received TAI UTC Delta Set message

Public Members

uint16_t **tai_utc_delta_new**
   Upcoming difference between TAI and UTC in seconds

uint16_t **padding**
   Always 0b0. Other values are Prohibited.

uint8_t **tai_delta_change**[5]
   TAI Seconds time of the upcoming TAI-UTC Delta change

struct esp_ble_mesh_server_recv_time_role_set_t
   Context of the received Time Role Set message
**Public Members**

`uint8_t time_role`

The Time Role for the element

`struct esp_ble_mesh_server_recv_scene_store_t`

Context of the received Scene Store message

**Public Members**

`uint16_t scene_number`

The number of scenes to be stored

`struct esp_ble_mesh_server_recv_scene_recall_t`

Context of the received Scene Recall message

**Public Members**

`bool op_en`

Indicate if optional parameters are included

`uint16_t scene_number`

The number of scenes to be recalled

`uint8_t tid`

Transaction ID

`uint8_t trans_time`

Time to complete state transition (optional)

`uint8_t delay`

Indicate message execution delay (C.1)

`struct esp_ble_mesh_server_recv_scene_delete_t`

Context of the received Scene Delete message

**Public Members**

`uint16_t scene_number`

The number of scenes to be deleted

`struct esp_ble_mesh_server_recv_scheduler_act_set_t`

Context of the received Scheduler Action Set message
Public Members

\texttt{\textbf{uint64_t index}}

Index of the Schedule Register entry to set

\texttt{\textbf{uint64_t year}}

Scheduled year for the action

\texttt{\textbf{uint64_t month}}

Scheduled month for the action

\texttt{\textbf{uint64_t day}}

Scheduled day of the month for the action

\texttt{\textbf{uint64_t hour}}

Scheduled hour for the action

\texttt{\textbf{uint64_t minute}}

Scheduled minute for the action

\texttt{\textbf{uint64_t second}}

Scheduled second for the action

\texttt{\textbf{uint64_t day_of_week}}

Scheduled day of the week for the action

\texttt{\textbf{uint64_t action}}

Action to be performed at the scheduled time

\texttt{\textbf{uint64_t trans_time}}

Transition time for this action

\texttt{\textbf{uint16_t scene_number}}

Scene number to be used for some actions

\texttt{\textbf{struct esp_ble_mesh_server_recv_time_status_t}}

Context of the received Time Status message

Public Members

\texttt{\textbf{uint8_t tai_seconds[5]}}

The current TAI time in seconds

\texttt{\textbf{uint8_t subsecond}}

The sub-second time in units of 1/256 second

\texttt{\textbf{uint8_t uncertainty}}

The estimated uncertainty in 10-millisecond steps
uint16_t time_authority
0 = No Time Authority, 1 = Time Authority

uint16_t tai_utc_delta
Current difference between TAI and UTC in seconds

uint8_t time_zone_offset
The local timezone offset in 15-minute increments

struct esp_ble_mesh_time_scene_server_cb_param_t
Time Scene Server Model callback parameters

Public Members

esp_ble_mesh_model_t *model
Pointer to Time and Scenes Server Models

esp_ble_mesh_msg_ctx_t ctx
Context of the received messages

esp_ble_mesh_time_scene_server_cb_value_t value
Value of the received Time and Scenes Messages

Macros
ESP_BLE_MESH_MODEL_TIME_CLI(cli_pub, cli_data)
Define a new Time Client Model.

备注: This API needs to be called for each element on which the application needs to have a Time Client Model.

参数
• cli_pub – Pointer to the unique struct esp_ble_mesh_model_pub_t.
• cli_data – Pointer to the unique struct esp_ble_mesh_client_t.
返回 New Time Client Model instance.

ESP_BLE_MESH_MODEL_SCENE_CLI(cli_pub, cli_data)
Define a new Scene Client Model.

备注: This API needs to be called for each element on which the application needs to have a Scene Client Model.

参数
• cli_pub – Pointer to the unique struct esp_ble_mesh_model_pub_t.
• cli_data – Pointer to the unique struct esp_ble_mesh_client_t.
返回 New Scene Client Model instance.
**Chapter 2. API 参考**

**ESP_BLE_MESH_MODEL_SCHEDULER_CLI (cli_pub, cli_data)**

Define a new Scheduler Client Model.

**备注**: This API needs to be called for each element on which the application needs to have a Scheduler Client Model.

**参数**
- `cli_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `cli_data` - Pointer to the unique struct `esp_ble_mesh_client_t`.

**返回**: New Scheduler Client Model instance.

**ESP_BLE_MESH_MODEL_TIME_SRV (srv_pub, srv_data)**

Time Scene Server Models related context.

Define a new Time Server Model.

**备注**: 1. The Time Server model is a root model. When this model is present on an Element, the corresponding Time Setup Server model shall also be present.
   - a. This model shall support model publication and model subscription.

**参数**
- `srv_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_time_srv_t`.

**返回**: New Time Server Model instance.

**ESP_BLE_MESH_MODEL_TIME_SETUP_SRV (srv_data)**

Define a new Time Setup Server Model.

**备注**: 1. The Time Setup Server model extends the Time Server model. Time is sensitive information that is propagated across a mesh network.
   - a. Only an authorized Time Client should be allowed to change the Time and Time Role states. A dedicated application key Bluetooth SIG Proprietary should be used on the Time Setup Server to restrict access to the server to only authorized Time Clients.
   - b. This model does not support subscribing nor publishing.

**参数**
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_time_setup_srv_t`.

**返回**: New Time Setup Server Model instance.

**ESP_BLE_MESH_MODEL_SCENE_SRV (srv_pub, srv_data)**

Define a new Scene Server Model.

**备注**: 1. The Scene Server model is a root model. When this model is present on an Element, the corresponding Scene Setup Server model shall also be present.
   - a. This model shall support model publication and model subscription.
   - b. The model may be present only on the Primary element of a node.

**参数**
- `srv_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_scene_srv_t`.

Submit Document Feedback
Chapter 2. API

**ESP_BLE_MESH_MODEL_SCENE_SETUP_SRV** (srv_pub, srv_data)
Define a new Scene Setup Server Model.

**备注**: 1. The Scene Setup Server model extends the Scene Server model and the Generic Default Transition Time Server model.
   a. This model shall support model subscription.
   b. The model may be present only on the Primary element of a node.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_scene_setup_srv_t`.

**返回** New Scene Setup Server Model instance.

**ESP_BLE_MESH_MODEL_SCHEDULER_SRV** (srv_pub, srv_data)
Define a new Scheduler Server Model.

**备注**: 1. The Scheduler Server model extends the Scene Server model. When this model is present on an Element, the corresponding Scheduler Setup Server model shall also be present.
   a. This model shall support model publication and model subscription.
   b. The model may be present only on the Primary element of a node.
   c. The model requires the Time Server model shall be present on the element.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_scheduler_srv_t`.

**返回** New Scheduler Server Model instance.

**ESP_BLE_MESH_MODEL_SCHEDULER_SETUP_SRV** (srv_pub, srv_data)
Define a new Scheduler Setup Server Model.

**备注**: 1. The Scheduler Setup Server model extends the Scheduler Server and the Scene Setup Server models.
   a. This model shall support model subscription.
   b. The model may be present only on the Primary element of a node.

**参数**
- **srv_pub** – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** – Pointer to the unique struct `esp_ble_mesh_scheduler_setup_srv_t`.

**返回** New Scheduler Setup Server Model instance.

**ESP_BLE_MESH_UNKNOWN_TAI_SECONDS**
Unknown TAI Seconds

**ESP_BLE_MESH_UNKNOWN_TAI_ZONE_CHANGE**
Unknown TAI of Zone Change

**ESP_BLE_MESH_UNKNOWN_TAI_DELTA_CHANGE**
Unknown TAI of Delta Change
**ESP_BLE_MESH_TAI_UTC_DELTA_MAX_VALUE**
Maximum TAI-UTC Delta value

**ESP_BLE_MESH_TAI_SECONDS_LEN**
Length of TAI Seconds

**ESP_BLE_MESH_TAI_OF_ZONE_CHANGE_LEN**
Length of TAI of Zone Change

**ESP_BLE_MESH_TAI_OF_DELTA_CHANGE_LEN**
Length of TAI of Delta Change

**ESP_BLE_MESH_INVALID_SCENE_NUMBER**
Invalid Scene Number

**ESP_BLE_MESH_SCENE_NUMBER_LEN**
Length of the Scene Number

**ESP_BLE_MESH_SCHEDULE_YEAR_ANY_YEAR**
Any year of the Scheduled year

**ESP_BLE_MESH_SCHEDULE_DAY_ANY_DAY**
Any day of the Scheduled day

**ESP_BLE_MESH_SCHEDULE_HOUR_ANY_HOUR**
Any hour of the Scheduled hour

**ESP_BLE_MESH_SCHEDULE_HOUR_ONCE_A_DAY**
Any hour of the Scheduled Day

**ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_HOUR**
Any minute of the Scheduled hour

**ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_MIN**
Every 15 minutes of the Scheduled hour

**ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_MIN**
Every 20 minutes of the Scheduled hour

**ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_HOUR**
Once of the Scheduled hour

**ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_MIN**
Any second of the Scheduled minute

**ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_SEC**
Every 15 seconds of the Scheduled minute
ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_SEC
Every 20 seconds of the Scheduled minute

ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_MIN
Once of the Scheduled minute

ESP_BLE_MESH_SCHEDULE_ACT_TURN_OFF
Scheduled Action - Turn Off

ESP_BLE_MESH_SCHEDULE_ACT_TURN_ON
Scheduled Action - Turn On

ESP_BLE_MESH_SCHEDULE_ACT_SCENE_RECALL
Scheduled Action - Scene Recall

ESP_BLE_MESH_SCHEDULE_ACT_NO_ACTION
Scheduled Action - No Action

ESP_BLE_MESH_SCHEDULE_SCENE_NO_SCENE
Scheduled Scene - No Scene

ESP_BLE_MESH_SCHEDULE_ENTRY_MAX_INDEX
Maximum number of Scheduled entries

ESP_BLE_MESH_TIME_NONE
Time Role - None

ESP_BLE_MESH_TIME_AUTHORITY
Time Role - Mesh Time Authority

ESP_BLE_MESH_TIME_RELAY
Time Role - Mesh Time Relay

ESP_BLE_MESH_TIME_CLI_NET
Time Role - Mesh Time Client

ESP_BLE_MESH_SCENE_SUCCESS
Scene operation - Success

ESP_BLE_MESH_SCENE_REG_FULL
Scene operation - Scene Register Full

ESP_BLE_MESH_SCENE_NOT_FOUND
Scene operation - Scene Not Found

Type Definitions
Chapter 2. API

```c
typedef void (*esp_ble_mesh_time_scene_client_cb_t)(
    esp_ble_mesh_time_scene_client_cb_event_t event,
    esp_ble_mesh_time_scene_client_cb_param_t *param)

Bluetooth Mesh Time Scene Client Model function.

Time Scene Client Model callback function type

Param event  Event type
Param param  Pointer to callback parameter
```

```c
typedef void (*esp_ble_mesh_time_scene_server_cb_t)(
    esp_ble_mesh_time_scene_server_cb_event_t event,
    esp_ble_mesh_time_scene_server_cb_param_t *param)

Bluetooth Mesh Time and Scenes Server Model function.

Time Scene Server Model callback function type

Param event  Event type
Param param  Pointer to callback parameter
```

**Enumerations**

```c
enum esp_ble_mesh_time_scene_client_cb_event_t

This enum value is the event of Time Scene Client Model
```

*Values:*

- `ESP_BLE_MESH_TIME_SCENE_CLIENT_GET_STATE_EVT`
- `ESP_BLE_MESH_TIME_SCENE_CLIENT_SET_STATE_EVT`
- `ESP_BLE_MESH_TIME_SCENE_CLIENT_PUBLISH_EVT`
- `ESP_BLE_MESH_TIME_SCENE_CLIENT_TIMEOUT_EVT`
- `ESP_BLE_MESH_TIME_SCENE_CLIENT_EVT_MAX`

```c
enum esp_ble_mesh_time_scene_server_cb_event_t

This enum value is the event of Time Scene Server Model
```

*Values:*

- `ESP_BLE_MESH_TIME_SCENE_SERVER_STATE_CHANGE_EVT`
  
  i. When `get_auto_rsp` is set to `ESP_BLE_MESH_SERVER_AUTO_RSP`, no event will be callback to the application layer when Time Scene Get messages are received.
  
  ii. When `set_auto_rsp` is set to `ESP_BLE_MESH_SERVER_AUTO_RSP`, this event will be callback to the application layer when Time Scene Set/Set Unack messages are received.

- `ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_GET_MSG_EVT`
  
  When `get_auto_rsp` is set to `ESP_BLE_MESH_SERVER_RSP_BY_APP`, this event will be callback to the application layer when Time Scene Get messages are received.

- `ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_SET_MSG_EVT`
  
  When `set_auto_rsp` is set to `ESP_BLE_MESH_SERVER_RSP_BY_APP`, this event will be callback to the application layer when Time Scene Set/Set Unack messages are received.
When status_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Time Status message is received.

enumerator **ESP_BLE_MESH_TIME_SCENE_SERVER_EVT_MAX**

Lighting Client/Server Models

Header File
- components/bt/esp_ble_mesh/api/models/include/esp_ble_mesh_lighting_model_api.h

Functions

**esp_err_t esp_ble_mesh_register_light_client_callback**(esp_ble_mesh_light_client_cb_t callback)

Register BLE Mesh Light Client Model callback.

参数 `callback` [in] pointer to the callback function.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_light_client_get_state**(esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_light_client_get_state_t *get_state)

Get the value of Light Server Model states using the Light Client Model get messages.

备注：If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_light_message_opcode_t in esp_ble_mesh_defs.h

参数
- `params` [in] Pointer to BLE Mesh common client parameters.
- `get_state` [in] Pointer of light get message value. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_light_client_set_state**(esp_ble_mesh_client_common_param_t *params, esp_ble_mesh_light_client_set_state_t *set_state)

Set the value of Light Server Model states using the Light Client Model set messages.

备注：If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_light_message_opcode_t in esp_ble_mesh_defs.h

参数
- `params` [in] Pointer to BLE Mesh common client parameters.
- `set_state` [in] Pointer of light set message value. Shall not be set to NULL.

返回 ESP_OK on success or error code otherwise.

**esp_err_t esp_ble_mesh_register_lighting_server_callback**(esp_ble_mesh_lighting_server_cb_t callback)

Register BLE Mesh Lighting Server Model callback.

参数 `callback` [in] Pointer to the callback function.

返回 ESP_OK on success or error code otherwise.
Chapter 2. API Reference

Unions

union esp_ble_mesh_light_client_get_state_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Client Model get message union.

Public Members

esp_ble_mesh_light_lc_property_get_t lc_property_get
For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_GET

union esp_ble_mesh_light_client_set_state_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Client Model set message union.

Public Members

esp_ble_mesh_light_lightness_set_t lightness_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET_UNACK

esp_ble_mesh_light_lightness_linear_set_t lightness_linear_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET_UNACK

esp_ble_mesh_light_lightness_default_set_t lightness_default_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET_UNACK

esp_ble_mesh_light_lightness_range_set_t lightness_range_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET_UNACK

esp_ble_mesh_light_ctl_set_t ctl_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET_UNACK

esp_ble_mesh_light_ctl_temperature_set_t ctl_temperature_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET_UNACK

esp_ble_mesh_light_ctl_temperature_range_set_t ctl_temperature_range_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET_UNACK

esp_ble_mesh_light_ctl_default_set_t ctl_default_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET_UNACK

esp_ble_mesh_light_hsl_set_t hsl_set
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET_UNACK

union esp_ble_mesh_light_client_status_cb_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Client Model received message union.

**Public Members**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS
Chapter 2. API

```
esp_ble_mesh_light_lightness_linear_status_cb_t lightness_linear_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_STATUS

esp_ble_mesh_light_lightness_last_status_cb_t lightness_last_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_STATUS

esp_ble_mesh_light_lightness_default_status_cb_t lightness_default_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_STATUS

esp_ble_mesh_light_lightness_range_status_cb_t lightness_range_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_STATUS

esp_ble_mesh_light_ctl_status_cb_t ctl_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_STATUS

esp_ble_mesh_light_ctl_temperature_status_cb_t ctl_temperature_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_STATUS

esp_ble_mesh_light_ctl_temperature_range_status_cb_t ctl_temperature_range_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_STATUS

esp_ble_mesh_light_ctl_default_status_cb_t ctl_default_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_STATUS

esp_ble_mesh_light_hsl_status_cb_t hsl_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_STATUS

esp_ble_mesh_light_hsl_target_status_cb_t hsl_target_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_STATUS

esp_ble_mesh_light_hsl_hue_status_cb_t hsl_hue_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_STATUS

esp_ble_mesh_light_hsl_saturation_status_cb_t hsl_saturation_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_STATUS

esp_ble_mesh_light_hsl_default_status_cb_t hsl_default_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_STATUS

esp_ble_mesh_light_hsl_range_status_cb_t hsl_range_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_STATUS

esp_ble_mesh_light_xyl_status_cb_t xyl_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_STATUS

esp_ble_mesh_light_xyl_target_status_cb_t xyl_target_status
For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_STATUS
```
**Chapter 2. API Reference**

```c
typedef struct esp_ble_mesh_light_xyl_default_status_cb_t {
    void (*xyl_default_status)(uint8_t op_code);
} xyl_default_status_cb_t;
```

For `ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_STATUS`

```c
typedef struct esp_ble_mesh_light_xyl_range_status_cb_t {
    void (*xyl_range_status)(uint8_t op_code);
} xyl_range_status_cb_t;
```

For `ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_STATUS`

```c
typedef struct esp_ble_mesh_light_lc_mode_status_cb_t {
    void (*lc_mode_status)(uint8_t op_code);
} lc_mode_status_cb_t;
```

For `ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_STATUS`

```c
typedef struct esp_ble_mesh_light_lc_range_status_cb_t {
    void (*lc_range_status)(uint8_t op_code);
} lc_range_status_cb_t;
```

For `ESP_BLE_MESH_MODEL_OP_LIGHT_LC_RANGE_STATUS`

```c
typedef struct esp_ble_mesh_light_lc_property_status_cb_t {
    void (*lc_property_status)(uint8_t op_code);
} lc_property_status_cb_t;
```

For `ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_STATUS`

```c
typedef union esp_ble_mesh_lighting_server_state_change_t {
    esp_ble_mesh_state_change_light_lightness_set_t lightness_set;
    esp_ble_mesh_state_change_light_lightness_linear_set_t lightness_linear_set;
    esp_ble_mesh_state_change_light_lightness_default_set_t lightness_default_set;
    esp_ble_mesh_state_change_light_lightness_range_set_t lightness_range_set;
    esp_ble_mesh_state_change_light_ctl_set_t ctl_set;
    esp_ble_mesh_state_change_light_ctl_temperature_set_t ctl_temp_set;
    esp_ble_mesh_state_change_light_ctl_temperature_range_set_t ctl_temp_range_set;
    esp_ble_mesh_state_change_light_ctl_default_set_t ctl_default_set;
} LightingServerModelstatechangevalueunion.
```

**Public Members**

```c
typedef struct esp_ble_mesh_state_change_light_lightness_set_t {
    void (*lightness_set)(uint8_t op_code);
} lightness_set_cb_t;
```

Light Lightness Set

```c
typedef struct esp_ble_mesh_state_change_light_lightness_linear_set_t {
    void (*lightness_linear_set)(uint8_t op_code);
} lightness_linear_set_cb_t;
```

Light Lightness Linear Set

```c
typedef struct esp_ble_mesh_state_change_light_lightness_default_set_t {
    void (*lightness_default_set)(uint8_t op_code);
} lightness_default_set_cb_t;
```

Light Lightness Default Set

```c
typedef struct esp_ble_mesh_state_change_light_lightness_range_set_t {
    void (*lightness_range_set)(uint8_t op_code);
} lightness_range_set_cb_t;
```

Light Lightness Range Set

```c
typedef struct esp_ble_mesh_state_change_light_ctl_set_t {
    void (*ctl_set)(uint8_t op_code);
} ctl_set_cb_t;
```

Light CTL Set

```c
typedef struct esp_ble_mesh_state_change_light_ctl_temperature_set_t {
    void (*ctl_temp_set)(uint8_t op_code);
} ctl_temp_set_cb_t;
```

Light CTL Temperature Set

```c
typedef struct esp_ble_mesh_state_change_light_ctl_temperature_range_set_t {
    void (*ctl_temp_range_set)(uint8_t op_code);
} ctl_temp_range_set_cb_t;
```

Light CTL Temperature Range Set

```c
typedef struct esp_ble_mesh_state_change_light_ctl_default_set_t {
    void (*ctl_default_set)(uint8_t op_code);
} ctl_default_set_cb_t;
```

Light CTL Default Set
**Chapter 2. API**

```c
esp_ble_mesh_state_change_light_hsl_set_t hsl_set
Light HSL Set

esp_ble_mesh_state_change_light_hsl_hue_set_t hsl_hue_set
Light HSL Hue Set

esp_ble_mesh_state_change_light_hsl_saturation_set_t hsl_saturation_set
Light HSL Saturation Set

esp_ble_mesh_state_change_light_hsl_default_set_t hsl_default_set
Light HSL Default Set

esp_ble_mesh_state_change_light_hsl_range_set_t hsl_range_set
Light HSL Range Set

esp_ble_mesh_state_change_light_xyl_set_t xyl_set
Light xyL Set

esp_ble_mesh_state_change_light_xyl_default_set_t xyl_default_set
Light xyL Default Set

esp_ble_mesh_state_change_light_xyl_range_set_t xyl_range_set
Light xyL Range Set

esp_ble_mesh_state_change_light_lc_mode_set_t lc_mode_set
Light LC Mode Set

esp_ble_mesh_state_change_light_lc_om_set_t lc_om_set
Light LC Occupancy Mode Set

esp_ble_mesh_state_change_light_lc_light_onoff_set_t lc_light_onoff_set
Light LC Light OnOff Set

esp_ble_mesh_state_change_light_lc_property_set_t lc_property_set
Light LC Property Set

esp_ble_mesh_state_change_sensor_status_t sensor_status
Sensor Status
```

union esp_ble_mesh_lighting_server_recv_get_msg_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model received get message union.

### Public Members

```c
esp_ble_mesh_server_recv_light_lc_property_get_t lc_property
Light LC Property Get
```

union esp_ble_mesh_lighting_server_recv_set_msg_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model received set message union.
Public Members

```c
esp_ble_mesh_server_recv_light_lightness_set_t lightness
Light Lightness Set/Light Lightness Set Unack
```

```c
esp_ble_mesh_server_recv_light_lightness_linear_set_t lightness_linear
Light Lightness Linear Set/Light Lightness Linear Set Unack
```

```c
esp_ble_mesh_server_recv_light_lightness_default_set_t lightness_default
Light Lightness Default Set/Light Lightness Default Set Unack
```

```c
esp_ble_mesh_server_recv_light_lightness_range_set_t lightness_range
Light Lightness Range Set/Light Lightness Range Set Unack
```

```c
esp_ble_mesh_server_recv_light_ctl_set_t ctl
Light CTL Set/Light CTL Set Unack
```

```c
esp_ble_mesh_server_recv_light_ctl_temperature_set_t ctl_temp
Light CTL Temperature Set/Light CTL Temperature Set Unack
```

```c
esp_ble_mesh_server_recv_light_ctl_temperature_range_set_t ctl_temp_range
Light CTL Temperature Range Set/Light CTL Temperature Range Set Unack
```

```c
esp_ble_mesh_server_recv_light_ctl_default_set_t ctl_default
Light CTL Default Set/Light CTL Default Set Unack
```

```c
esp_ble_mesh_server_recv_light_hsl_set_t hsl
Light HSL Set/Light HSL Set Unack
```

```c
esp_ble_mesh_server_recv_light_hsl_hue_set_t hsl_hue
Light HSL Hue Set/Light HSL Hue Set Unack
```

```c
esp_ble_mesh_server_recv_light_hsl_saturation_set_t hsl_saturation
Light HSL Saturation Set/Light HSL Saturation Set Unack
```

```c
esp_ble_mesh_server_recv_light_hsl_default_set_t hsl_default
Light HSL Default Set/Light HSL Default Set Unack
```

```c
esp_ble_mesh_server_recv_light_hsl_range_set_t hsl_range
Light HSL Range Set/Light HSL Range Set Unack
```

```c
esp_ble_mesh_server_recv_light_xyl_set_t xyl
Light xyL Set/Light xyL Set Unack
```

```c
esp_ble_mesh_server_recv_light_xyl_default_set_t xyl_default
Light xyL Default Set/Light xyL Default Set Unack
```

```c
esp_ble_mesh_server_recv_light_xyl_range_set_t xyl_range
Light xyL Range Set/Light xyL Range Set Unack
```
Chapter 2. API

```
#include <esp_ble_mesh_lighting_model_api.h>
union esp_ble_mesh_lighting_server_recv_status_msg_t
    #include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model received status message union.

Public Members

```

```
#include <esp_ble_mesh_lighting_model_api.h>
union esp_ble_mesh_lighting_server_cb_value_t
    #include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model callback value union.

Public Members

```

```
#include <esp_ble_mesh_lighting_model_api.h>
union esp_ble_mesh_lighting_server_state_change_t
    ESP_BLE_MESH_LIGHTING_SERVER_STATE_CHANGE_EVT

Public Members

```

```
#include <esp_ble_mesh_lighting_model_api.h>
union esp_ble_mesh_lighting_server_recv_get_msg_t
    ESP_BLE_MESH_LIGHTING_SERVER_RECV_GET_MSG_EVT

Public Members

```

```
#include <esp_ble_mesh_lighting_model_api.h>
union esp_ble_mesh_lighting_server_recv_set_msg_t
    ESP_BLE_MESH_LIGHTING_SERVER_RECV_SET_MSG_EVT

Public Members

```

```
#include <esp_ble_mesh_lighting_model_api.h>
union esp_ble_mesh_lighting_server_recv_status_msg_t
    ESP_BLE_MESH_LIGHTING_SERVER_RECV_STATUS_MSG_EVT

Structures

```
struct esp_ble_mesh_light_lightness_set_t
    Bluetooth Mesh Light Lightness Client Model Get and Set parameters structure.
    Parameters of Light Lightness Set

Public Members
```
bool op_en
   Indicate if optional parameters are included

uint16_t lightness
   Target value of lightness actual state

uint8_t tid
   Transaction ID

uint8_t trans_time
   Time to complete state transition (optional)

uint8_t delay
   Indicate message execution delay (C.1)

struct esp_ble_mesh_light_lightness_linear_set_t
   Parameters of Light Lightness Linear Set

Public Members

bool op_en
   Indicate if optional parameters are included

uint16_t lightness
   Target value of lightness linear state

uint8_t tid
   Transaction ID

uint8_t trans_time
   Time to complete state transition (optional)

uint8_t delay
   Indicate message execution delay (C.1)

struct esp_ble_mesh_light_lightness_default_set_t
   Parameter of Light Lightness Default Set

Public Members

uint16_t lightness
   The value of the Light Lightness Default state

struct esp_ble_mesh_light_lightness_range_set_t
   Parameters of Light Lightness Range Set
Public Members

```c
uint16_t range_min
Value of range min field of light lightness range state
```

```c
uint16_t range_max
Value of range max field of light lightness range state
```

```c
struct esp_ble_mesh_light_ctl_set_t
Parameters of Light CTL Set
```

Public Members

```c
bool op_en
Indicate if optional parameters are included
```

```c
uint16_t ctl_lightness
Target value of light ctl lightness state
```

```c
uint16_t ctl_temperatrue
Target value of light ctl temperature state
```

```c
int16_t ctl_delta_uv
Target value of light ctl delta UV state
```

```c
uint8_t tid
Transaction ID
```

```c
uint8_t trans_time
Time to complete state transition (optional)
```

```c
uint8_t delay
Indicate message execution delay (C.1)
```

```c
struct esp_ble_mesh_light_ctl_temperature_set_t
Parameters of Light CTL Temperature Set
```

Public Members

```c
bool op_en
Indicate if optional parameters are included
```

```c
uint16_t ctl_temperatrue
Target value of light ctl temperature state
```

```c
int16_t ctl_delta_uv
Target value of light ctl delta UV state
```
uint8_t tid
Transaction ID

uint8_t trans_time
Time to complete state transition (optional)

uint8_t delay
Indicate message execution delay (C.1)

struct esp_ble_mesh_light_ctl_temperature_range_set_t
Parameters of Light CTL Temperature Range Set

**Public Members**

uint16_t range_min
Value of temperature range min field of light ctl temperature range state

uint16_t range_max
Value of temperature range max field of light ctl temperature range state

struct esp_ble_mesh_light_ctl_default_set_t
Parameters of Light CTL Default Set

**Public Members**

uint16_t lightness
Value of light lightness default state

uint16_t temperature
Value of light temperature default state

int16_t delta_uv
Value of light delta UV default state

struct esp_ble_mesh_light_hsl_set_t
Parameters of Light HSL Set

**Public Members**

bool op_en
Indicate if optional parameters are included

uint16_t hsl_lightness
Target value of light hsl lightness state
uint16_t hsl_hue
    Target value of light hsl hue state

uint16_t hsl_saturation
    Target value of light hsl saturation state

uint8_t tid
    Transaction ID

uint8_t trans_time
    Time to complete state transition (optional)

uint8_t delay
    Indicate message execution delay (C.1)

struct esp_ble_mesh_light_hsl_hue_set_t
    Parameters of Light HSL Hue Set

    Public Members

    bool op_en
        Indicate if optional parameters are included

    uint16_t hue
        Target value of light hsl hue state

    uint8_t tid
        Transaction ID

    uint8_t trans_time
        Time to complete state transition (optional)

    uint8_t delay
        Indicate message execution delay (C.1)

struct esp_ble_mesh_light_hsl_saturation_set_t
    Parameters of Light HSL Saturation Set

    Public Members

    bool op_en
        Indicate if optional parameters are included

    uint16_t saturation
        Target value of light hsl hue state
uint8_t **tid**
  Transaction ID

uint8_t **trans_time**
  Time to complete state transition (optional)

uint8_t **delay**
  Indicate message execution delay (C.1)

**struct esp_ble_mesh_light_hsl_default_set_t**
  Parameters of Light HSL Default Set

**Public Members**

uint16_t **lightness**
  Value of light lightness default state

uint16_t **hue**
  Value of light hue default state

uint16_t **saturation**
  Value of light saturation default state

**struct esp_ble_mesh_light_hsl_range_set_t**
  Parameters of Light HSL Range Set

**Public Members**

uint16_t **hue_range_min**
  Value of hue range min field of light hsl hue range state

uint16_t **hue_range_max**
  Value of hue range max field of light hsl hue range state

uint16_t **saturation_range_min**
  Value of saturation range min field of light hsl saturation range state

uint16_t **saturation_range_max**
  Value of saturation range max field of light hsl saturation range state

**struct esp_ble_mesh_light_xyl_set_t**
  Parameters of Light xyL Set

**Public Members**
bool op_en
   Indicate whether optional parameters included

uint16_t xyl_lightness
   The target value of the Light xyL Lightness state

uint16_t xyl_x
   The target value of the Light xyL x state

uint16_t xyl_y
   The target value of the Light xyL y state

uint8_t tid
   Transaction Identifier

uint8_t trans_time
   Time to complete state transition (optional)

uint8_t delay
   Indicate message execution delay (C.1)

struct esp_ble_mesh_light_xyl_default_set_t
   Parameters of Light xyL Default Set

Public Members

uint16_t lightness
   The value of the Light Lightness Default state

uint16_t xyl_x
   The value of the Light xyL x Default state

uint16_t xyl_y
   The value of the Light xyL y Default state

struct esp_ble_mesh_light_xyl_range_set_t
   Parameters of Light xyL Range Set

Public Members

uint16_t xyl_x_range_min
   The value of the xyL x Range Min field of the Light xyL x Range state

uint16_t xyl_x_range_max
   The value of the xyL x Range Max field of the Light xyL x Range state
uint16_t xyl_y_range_min
The value of the xyl y Range Min field of the Light xyl y Range state

uint16_t xyl_y_range_max
The value of the xyl y Range Max field of the Light xyl y Range state

struct esp_ble_mesh_light_lc_mode_set_t
Parameter of Light LC Mode Set

Public Members

uint8_t mode
The target value of the Light LC Mode state

struct esp_ble_mesh_light_lc_om_set_t
Parameter of Light LC OM Set

Public Members

uint8_t mode
The target value of the Light LC Occupancy Mode state

struct esp_ble_mesh_light_lc_light_onoff_set_t
Parameters of Light LC Light OnOff Set

Public Members

bool op_en
Indicate whether optional parameters included

uint8_t light_onoff
The target value of the Light LC Light OnOff state

uint8_t tid
Transaction Identifier

uint8_t trans_time
Time to complete state transition (optional)

uint8_t delay
Indicate message execution delay (C.1)

struct esp_ble_mesh_light_lc_property_get_t
Parameter of Light LC Property Get
Public Members

```c
uint16_t property_id
  Property ID identifying a Light LC Property
```

```c
struct esp_ble_mesh_light_lc_property_set_t
  Parameters of Light LC Property Set
```

Public Members

```c
uint16_t property_id
  Property ID identifying a Light LC Property
```

```c
struct net_buf_simple *property_value
  Raw value for the Light LC Property
```

```c
struct esp_ble_mesh_light_lightness_status_cb_t
  Bluetooth Mesh Light Lightness Client Model Get and Set callback parameters structure.
  Parameters of Light Lightness Status
```

Public Members

```c
bool op_en
  Indicate if optional parameters are included
```

```c
uint16_t present_lightness
  Current value of light lightness actual state
```

```c
uint16_t target_lightness
  Target value of light lightness actual state (optional)
```

```c
uint8_t remain_time
  Time to complete state transition (C.1)
```

```c
struct esp_ble_mesh_light_lightness_linear_status_cb_t
  Parameters of Light Lightness Linear Status
```

Public Members

```c
bool op_en
  Indicate if optional parameters are included
```

```c
uint16_t present_lightness
  Current value of light lightness linear state
```
Chapter 2. API

uint16_t target_lightness
   Target value of light lightness linear state (optional)

uint8_t remain_time
   Time to complete state transition (C.1)

struct esp_ble_mesh_light_lightness_last_status_cb_t
   Parameter of Light Lightness Last Status

Public Members

uint16_t lightness
   The value of the Light Lightness Last state

struct esp_ble_mesh_light_lightness_default_status_cb_t
   Parameter of Light Lightness Default Status

Public Members

uint16_t lightness
   The value of the Light Lightness default State

struct esp_ble_mesh_light_lightness_range_status_cb_t
   Parameters of Light Lightness Range Status

Public Members

uint8_t status_code
   Status Code for the request message

uint16_t range_min
   Value of range min field of light lightness range state

uint16_t range_max
   Value of range max field of light lightness range state

struct esp_ble_mesh_light_ctl_status_cb_t
   Parameters of Light CTL Status

Public Members

bool op_en
   Indicate if optional parameters are included

uint16_t present_ctl_lightness
   Current value of light ctl lightness state
uint16_t `present_ctl_temperature`
   Current value of light ctl temperature state

uint16_t `target_ctl_lightness`
   Target value of light ctl lightness state (optional)

uint16_t `target_ctl_temperature`
   Target value of light ctl temperature state (C.1)

uint8_t `remain_time`
   Time to complete state transition (C.1)

struct `esp_ble_mesh_light_ctl_temperature_status_cb_t`
   Parameters of Light CTL Temperature Status

   **Public Members**

   bool `op_en`
      Indicate if optional parameters are included

   uint16_t `present_ctl_temperature`
      Current value of light ctl temperature state

   uint16_t `present_ctl_delta_uv`
      Current value of light ctl delta UV state

   uint16_t `target_ctl_temperature`
      Target value of light ctl temperature state (optional)

   uint16_t `target_ctl_delta_uv`
      Target value of light ctl delta UV state (C.1)

   uint8_t `remain_time`
      Time to complete state transition (C.1)

struct `esp_ble_mesh_light_ctl_temperature_range_status_cb_t`
   Parameters of Light CTL Temperature Range Status

   **Public Members**

   uint8_t `status_code`
      Status code for the request message

   uint16_t `range_min`
      Value of temperature range min field of light ctl temperature range state
uint16_t range_max
Value of temperature range max field of light ctl temperature range state

struct esp_ble_mesh_light_ctl_default_status_cb_t
Parameters of Light CTL Default Status

Public Members

uint16_t lightness
Value of light lightness default state

uint16_t temperature
Value of light temperature default state

int16_t delta_uv
Value of light delta UV default state

struct esp_ble_mesh_light_hsl_status_cb_t
Parameters of Light HSL Status

Public Members

bool op_en
Indicate if optional parameters are included

uint16_t hsl_lightness
Current value of light hsl lightness state

uint16_t hsl_hue
Current value of light hsl hue state

uint16_t hsl_saturation
Current value of light hsl saturation state

uint8_t remain_time
Time to complete state transition (optional)

struct esp_ble_mesh_light_hsl_target_status_cb_t
Parameters of Light HSL Target Status

Public Members

bool op_en
Indicate if optional parameters are included
uint16_t hsl_lightness_target
    Target value of light hsl lightness state

uint16_t hsl_hue_target
    Target value of light hsl hue state

uint16_t hsl_saturation_target
    Target value of light hsl saturation state

uint8_t remain_time
    Time to complete state transition (optional)

struct esp_ble_mesh_light_hsl_hue_status_cb_t
    Parameters of Light HSL Hue Status

    Public Members

    bool op_en
        Indicate if optional parameters are included

    uint16_t present_hue
        Current value of light hsl hue state

    uint16_t target_hue
        Target value of light hsl hue state (optional)

    uint8_t remain_time
        Time to complete state transition (C.1)

struct esp_ble_mesh_light_hsl_saturation_status_cb_t
    Parameters of Light HSL Saturation Status

    Public Members

    bool op_en
        Indicate if optional parameters are included

    uint16_t present_saturation
        Current value of light hsl saturation state

    uint16_t target_saturation
        Target value of light hsl saturation state (optional)

    uint8_t remain_time
        Time to complete state transition (C.1)

struct esp_ble_mesh_light_hsl_default_status_cb_t
    Parameters of Light HSL Default Status
Public Members

```c
uint16_t lightness
  Value of light lightness default state

uint16_t hue
  Value of light hue default state

uint16_t saturation
  Value of light saturation default state
```

```c
struct esp_ble_mesh_light_hsl_range_status_cb_t
  Parameters of Light HSL Range Status
```

Public Members

```c
uint8_t status_code
  Status code for the request message

uint16_t hue_range_min
  Value of hue range min field of light hsl hue range state

uint16_t hue_range_max
  Value of hue range max field of light hsl hue range state

uint16_t saturation_range_min
  Value of saturation range min field of light hsl saturation range state

uint16_t saturation_range_max
  Value of saturation range max field of light hsl saturation range state
```

```c
struct esp_ble_mesh_light_xyl_status_cb_t
  Parameters of Light xyL Status
```

Public Members

```c
bool op_en
  Indicate whether optional parameters included

uint16_t xyl_lightness
  The present value of the Light xyL Lightness state

uint16_t xyl_x
  The present value of the Light xyL x state

uint16_t xyl_y
  The present value of the Light xyL y state
```
uint8_t \texttt{remain\_time}

Time to complete state transition (optional)

\begin{verbatim}
struct esp_ble_mesh_light_xyl_target_status_cb_t
Parameters of Light \text{xyL Target Status}

\textbf{Public Members}

bool \texttt{op\_en}

Indicate whether optional parameters included

uint16_t \texttt{target\_xyl\_lightness}

The target value of the Light \text{xyL Lightness} state

uint16_t \texttt{target\_xyl\_x}

The target value of the Light \text{xyL x} state

uint16_t \texttt{target\_xyl\_y}

The target value of the Light \text{xyL y} state

uint8_t \texttt{remain\_time}

Time to complete state transition (optional)
\end{verbatim}

\begin{verbatim}
struct esp_ble_mesh_light_xyl_default_status_cb_t
Parameters of Light \text{xyL Default Status}

\textbf{Public Members}

uint16_t \texttt{lightness}

The value of the Light \text{Lightness Default} state

uint16_t \texttt{xyl\_x}

The value of the Light \text{xyL x Default} state

uint16_t \texttt{xyl\_y}

The value of the Light \text{xyL y Default} state
\end{verbatim}

\begin{verbatim}
struct esp_ble_mesh_light_xyl_range_status_cb_t
Parameters of Light \text{xyL Range Status}

\textbf{Public Members}

uint8_t \texttt{status\_code}

Status Code for the requesting message
\end{verbatim}
Chapter 2. API

```c
uint16_t xyl_x_range_min
    The value of the xyL x Range Min field of the Light xyL x Range state

uint16_t xyl_x_range_max
    The value of the xyL x Range Max field of the Light xyL x Range state

uint16_t xyl_y_range_min
    The value of the xyL y Range Min field of the Light xyL y Range state

uint16_t xyl_y_range_max
    The value of the xyL y Range Max field of the Light xyL y Range state
```

```c
struct esp_ble_mesh_light_lc_mode_status_cb_t
    Parameter of Light LC Mode Status

Public Members

uint8_t mode
    The present value of the Light LC Mode state
```

```c
struct esp_ble_mesh_light_lc_om_status_cb_t
    Parameter of Light LC OM Status

Public Members

uint8_t mode
    The present value of the Light LC Occupancy Mode state
```

```c
struct esp_ble_mesh_light_lc_light_onoff_status_cb_t
    Parameters of Light LC Light OnOff Status

Public Members

bool op_en
    Indicate whether optional parameters included

uint8_t present_light_onoff
    The present value of the Light LC Light OnOff state

uint8_t target_light_onoff
    The target value of the Light LC Light OnOff state (Optional)

uint8_t remain_time
    Time to complete state transition (C.1)
```

```c
struct esp_ble_mesh_light_lc_property_status_cb_t
    Parameters of Light LC Property Status
```
Chapter 2. API 参考

Public Members

uint16_t **property_id**
Property ID identifying a Light LC Property

struct net_buf_simple **property_value**
Raw value for the Light LC Property

struct esp_ble_mesh_light_client_cb_param_t
Lighting Client Model callback parameters

Public Members

int **error_code**
Appropriate error code

*esp_ble_mesh_client_common_param_t* **params**
The client common parameters.

*esp_ble_mesh_light_client_status_cb_t* **status_cb**
The light status message callback values

struct esp_ble_mesh_light_lightness_state_t
Parameters of Light Lightness state

Public Members

uint16_t **lightness_linear**
The present value of Light Lightness Linear state

uint16_t **target_lightness_linear**
The target value of Light Lightness Linear state

uint16_t **lightness_actual**
The present value of Light Lightness Actual state

uint16_t **target_lightness_actual**
The target value of Light Lightness Actual state

uint16_t **lightness_last**
The value of Light Lightness Last state

uint16_t **lightness_default**
The value of Light Lightness Default state

uint8_t **status_code**
The status code of setting Light Lightness Range state
\begin{verbatim}
uint16_t lightness_range_min
    The minimum value of Light Lightness Range state

uint16_t lightness_range_max
    The maximum value of Light Lightness Range state

struct esp_ble_mesh_light_lightness_srv_t
    User data of Light Lightness Server Model

Public Members

esp_ble_mesh_model_t *model
    Pointer to the Lighting Lightness Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl
    Response control of the server model received messages

esp_ble_mesh_light_lightness_state_t *state
    Parameters of the Light Lightness state

esp_ble_mesh_last_msg_info_t last
    Parameters of the last received set message

esp_ble_mesh_state_transition_t actual_transition
    Parameters of state transition

esp_ble_mesh_state_transition_t linear_transition
    Parameters of state transition

int32_t tt_delta_lightness_actual
    Delta change value of lightness actual state transition

int32_t tt_delta_lightness_linear
    Delta change value of lightness linear state transition

struct esp_ble_mesh_light_lightness_setup_srv_t
    User data of Light Lightness Setup Server Model

Public Members

esp_ble_mesh_model_t *model
    Pointer to the Lighting Lightness Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl
    Response control of the server model received messages
\end{verbatim}
**esp_ble_mesh_light_lightness_state_t** *state*
Parameters of the Light Lightness state

**struct esp_ble_mesh_light_ctl_state_t**
Parameters of Light CTL state

**Public Members**

uint16_t **lightness**
The present value of Light CTL Lightness state

uint16_t **target_lightness**
The target value of Light CTL Lightness state

uint16_t **temperature**
The present value of Light CTL Temperature state

uint16_t **target_temperature**
The target value of Light CTL Temperature state

int16_t **delta_uv**
The present value of Light CTL Delta UV state

int16_t **target_delta_uv**
The target value of Light CTL Delta UV state

uint8_t **status_code**
The status code of setting Light CTL Temperature Range state

uint16_t **temperature_range_min**
The minimum value of Light CTL Temperature Range state

uint16_t **temperature_range_max**
The maximum value of Light CTL Temperature Range state

uint16_t **lightness_default**
The value of Light Lightness Default state

uint16_t **temperature_default**
The value of Light CTL Temperature Default state

int16_t **delta_uv_default**
The value of Light CTL Delta UV Default state

**struct esp_ble_mesh_light_ctl_srv_t**
User data of Light CTL Server Model
Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Lighting CTL Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp ctrl`
Response control of the server model received messages

`esp_ble_mesh_light_ctl_state_t *state`
Parameters of the Light CTL state

`esp_ble_mesh_last_msg_info_t last`
Parameters of the last received set message

`esp_ble_mesh_state_transition_t transition`
Parameters of state transition

`int32_t tt_delta_lightness`
Delta change value of lightness state transition

`int32_t tt_delta_temperature`
Delta change value of temperature state transition

`int32_t tt_delta_delta_uv`
Delta change value of delta uv state transition

`struct esp_ble_mesh_light_ctl_temp_srv_t`
User data of Light CTL Temperature Server Model

Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Lighting CTL Setup Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp ctrl`
Response control of the server model received messages

`esp_ble_mesh_light_ctl_state_t *state`
Parameters of the Light CTL state

`struct esp_ble_mesh_light_ctl_temp_srv_t`
User data of Light CTL Temperature Server Model

Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Lighting CTL Temperature Server Model. Initialized internally.
**Chapter 2. API 参考**

- `esp_ble_mesh_server_rsp_ctrl_t` **rsp_ctrl**
  - Response control of the server model received messages

- `esp_ble_mesh_light_ctl_state_t` **state**
  - Parameters of the Light CTL state

- `esp_ble_mesh_last_msg_info_t` **last**
  - Parameters of the last received set message

- `esp_ble_mesh_state_transition_t` **transition**
  - Parameters of state transition

- `int32_t` **tt_delta_temperature**
  - Delta change value of temperature state transition

- `int32_t` **tt_delta_delta_uv**
  - Delta change value of delta uv state transition

- `struct esp_ble_mesh_light_hsl_state_t`
  - Parameters of Light HSL state

**Public Members**

- `uint16_t` **lightness**
  - The present value of Light HSL Lightness state

- `uint16_t` **target_lightness**
  - The target value of Light HSL Lightness state

- `uint16_t` **hue**
  - The present value of Light HSL Hue state

- `uint16_t` **target_hue**
  - The target value of Light HSL Hue state

- `uint16_t` **saturation**
  - The present value of Light HSL Saturation state

- `uint16_t` **target_saturation**
  - The target value of Light HSL Saturation state

- `uint16_t` **lightness_default**
  - The value of Light Lightness Default state

- `uint16_t` **hue_default**
  - The value of Light HSL Hue Default state
uint16_t saturation_default
The value of Light HSL Saturation Default state

uint8_t status_code
The status code of setting Light HSL Hue & Saturation Range state

uint16_t hue_range_min
The minimum value of Light HSL Hue Range state

uint16_t hue_range_max
The maximum value of Light HSL Hue Range state

uint16_t saturation_range_min
The minimum value of Light HSL Saturation state

uint16_t saturation_range_max
The maximum value of Light HSL Saturation state

struct esp_ble_mesh_light_hsl_srv_t
User data of Light HSL Server Model

Public Members

esp_ble_mesh_model_t *model
Pointer to the Lighting HSL Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl
Response control of the server model received messages

esp_ble_mesh_light_hsl_state_t *state
Parameters of the Light HSL state

esp_ble_mesh_last_msg_info_t last
Parameters of the last received set message

esp_ble_mesh_state_transition_t *transition
Parameters of state transition

int32_t tt_delta_lightness
Delta change value of lightness state transition

int32_t tt_delta_hue
Delta change value of hue state transition

int32_t tt_delta_saturation
Delta change value of saturation state transition

struct esp_ble_mesh_light_hsl_setup_srv_t
User data of Light HSL Setup Server Model
Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Lighting HSL Setup Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`esp_ble_mesh_light_hsl_state_t *state`
Parameters of the Light HSL state

`struct esp_ble_mesh_light_hsl_hue_srv_t`
User data of Light HSL Hue Server Model

Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Lighting HSL Hue Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`esp_ble_mesh_light_hsl_state_t *state`
Parameters of the Light HSL state

`esp_ble_mesh_last_msg_info_t last`
Parameters of the last received set message

`esp_ble_mesh_state_transition_t transition`
Parameters of state transition

`int32_t tt_delta_hue`
Delta change value of hue state transition

`struct esp_ble_mesh_light_hsl_sat_srv_t`
User data of Light HSL Saturation Server Model

Public Members

`esp_ble_mesh_model_t *model`
Pointer to the Lighting HSL Saturation Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`esp_ble_mesh_light_hsl_state_t *state`
Parameters of the Light HSL state
\textit{esp\_ble\_mesh\_last\_msg\_info\_t \texttt{last}}

Parameters of the last received set message

\textit{esp\_ble\_mesh\_state\_transition\_t \texttt{transition}}

Parameters of state transition

\texttt{int32\_t \texttt{tt\_delta\_saturation}}

Delta change value of saturation state transition

\texttt{struct \textit{esp\_ble\_mesh\_light\_xyl\_state\_t}}

Parameters of Light xyL state

\textbf{Public Members}

\texttt{uint16\_t \texttt{lightness}}

The present value of Light xyL Lightness state

\texttt{uint16\_t \texttt{target\_lightness}}

The target value of Light xyL Lightness state

\texttt{uint16\_t \texttt{x}}

The present value of Light xyL x state

\texttt{uint16\_t \texttt{target\_x}}

The target value of Light xyL x state

\texttt{uint16\_t \texttt{y}}

The present value of Light xyL y state

\texttt{uint16\_t \texttt{target\_y}}

The target value of Light xyL y state

\texttt{uint16\_t \texttt{lightness\_default}}

The value of Light Lightness Default state

\texttt{uint16\_t \texttt{x\_default}}

The value of Light xyL x Default state

\texttt{uint16\_t \texttt{y\_default}}

The value of Light xyL y Default state

\texttt{uint8\_t \texttt{status\_code}}

The status code of setting Light xyL x & y Range state

\texttt{uint16\_t \texttt{x\_range\_min}}

The minimum value of Light xyL x Range state
uint16_t \texttt{x\_range\_max}  
The maximum value of Light xyL x Range state

uint16_t \texttt{y\_range\_min}  
The minimum value of Light xyL y Range state

uint16_t \texttt{y\_range\_max}  
The maximum value of Light xyL y Range state

\textbf{Public Members}

\begin{verbatim}
\texttt{esp\_ble\_mesh\_model\_t *model}  
Pointer to the Lighting xyL Server Model. Initialized internally.

\texttt{esp\_ble\_mesh\_server\_rsp\_ctrl\_t *rsp\_ctrl}  
Response control of the server model received messages

\texttt{esp\_ble\_mesh\_light\_xyl\_state\_t *state}  
Parameters of the Light xyL state

\texttt{esp\_ble\_mesh\_last\_msg\_info\_t *last}  
Parameters of the last received set message

\texttt{esp\_ble\_mesh\_state\_transition\_t *transition}  
Parameters of state transition

\texttt{int32_t tt\_delta\_lightness}  
Delta change value of lightness state transition

\texttt{int32_t tt\_delta\_x}  
Delta change value of x state transition

\texttt{int32_t tt\_delta\_y}  
Delta change value of y state transition
\end{verbatim}

\textbf{struct \texttt{esp\_ble\_mesh\_light\_xyl\_setup\_srv\_t}}  
User data of Light xyL Setup Server Model

\textbf{Public Members}

\begin{verbatim}
\texttt{esp\_ble\_mesh\_model\_t *model}  
Pointer to the Lighting xyL Setup Server Model. Initialized internally.
\end{verbatim}
Chapter 2. API 参考

`esp_ble_mesh_server_rsp_ctrl_t *rsp_ctrl`
Response control of the server model received messages

`esp_ble_mesh_light_xyl_state_t *state`
Parameters of the Light xyl state

`struct esp_ble_mesh_light_lc_state_t`
Parameters of Light LC states

**Public Members**

`uint32_t mode`
0b0 The controller is turned off.
- The binding with the Light Lightness state is disabled. 0b1 The controller is turned on.
- The binding with the Light Lightness state is enabled. The value of Light LC Mode state

`uint32_t occupancy_mode`
The value of Light LC Occupancy Mode state

`uint32_t light_onoff`
The present value of Light LC Light OnOff state

`uint32_t target_light_onoff`
The target value of Light LC Light OnOff state

`uint32_t occupancy`
The value of Light LC Occupancy state

`uint32_t ambient_luxlevel`
The value of Light LC Ambient LuxLevel state

`uint16_t linear_output`
- Light LC Linear Output = max((Lightness Out)^2/65535, Regulator Output)
- If the Light LC Mode state is set to 0b1, the binding is enabled and upon a change of the Light LC Linear Output state, the following operation shall be performed: Light Lightness Linear = Light LC Linear Output
- If the Light LC Mode state is set to 0b0, the binding is disabled (i.e., upon a change of the Light LC Linear Output state, no operation on the Light Lightness Linear state is performed). The value of Light LC Linear Output state

`struct esp_ble_mesh_light_lc_property_state_t`
Parameters of Light Property states. The Light LC Property states are read / write states that determine the configuration of a Light Lightness Controller. Each state is represented by a device property and is controlled by Light LC Property messages.

**Public Members**
**uint32_t time_occupancy_delay**
A timing state that determines the delay for changing the Light LC Occupancy state upon receiving a Sensor Status message from an occupancy sensor. The value of Light LC Time Occupancy Delay state

**uint32_t time_fade_on**
A timing state that determines the time the controlled lights fade to the level determined by the Light LC Lightness On state. The value of Light LC Time Fade On state

**uint32_t time_run_on**
A timing state that determines the time the controlled lights stay at the level determined by the Light LC Lightness On state. The value of Light LC Time Run On state

**uint32_t time_fade**
A timing state that determines the time the controlled lights fade from the level determined by the Light LC Lightness On state to the level determined by the Light Lightness Prolong state. The value of Light LC Time Fade state

**uint32_t time_prolong**
A timing state that determines the time the controlled lights stay at the level determined by the Light LC Lightness Prolong state. The value of Light LC Time Prolong state

**uint32_t time_fade_standby_auto**
A timing state that determines the time the controlled lights fade from the level determined by the Light LC Lightness Prolong state to the level determined by the Light LC Lightness Standby state when the transition is automatic. The value of Light LC Time Fade Standby Auto state

**uint32_t time_fade_standby_manual**
A timing state that determines the time the controlled lights fade from the level determined by the Light LC Lightness Prolong state to the level determined by the Light LC Lightness Standby state when the transition is triggered by a change in the Light LC Light OnOff state. The value of Light LC Time Fade Standby Manual state

**uint16_t lightness_on**
A lightness state that determines the perceptive light lightness at the Occupancy and Run internal controller states. The value of Light LC Lightness On state

**uint16_t lightness_prolong**
A lightness state that determines the light lightness at the Prolong internal controller state. The value of Light LC Lightness Prolong state

**uint16_t lightness_standby**
A lightness state that determines the light lightness at the Standby internal controller state. The value of Light LC Lightness Standby state

**uint16_t ambient_luxlevel_on**
A uint16 state representing the Ambient LuxLevel level that determines if the controller transitions from the Light Control Standby state. The value of Light LC Ambient LuxLevel On state

**uint16_t ambient_luxlevel_prolong**
A uint16 state representing the required Ambient LuxLevel level in the Prolong state. The value of Light LC Ambient LuxLevel Prolong state
uint16_t ambient_luxlevel_standby
A uint16 state representing the required Ambient LuxLevel level in the Standby state. The value of Light LC Ambient LuxLevel Standby state

float regulator_kiu
A float32 state representing the integral coefficient that determines the integral part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel is less than LuxLevel Out. Valid range: 0.0 ~ 1000.0. The default value is 250.0. The value of Light LC Regulator Kiu state

float regulator_kid
A float32 state representing the integral coefficient that determines the integral part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel is greater than or equal to the value of the LuxLevel Out state. Valid range: 0.0 ~ 1000.0. The default value is 25.0. The value of Light LC Regulator Kid state

float regulator_kpu
A float32 state representing the proportional coefficient that determines the proportional part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel is less than the value of the LuxLevel Out state. Valid range: 0.0 ~ 1000.0. The default value is 80.0. The value of Light LC Regulator Kpu state

float regulator_kpd
A float32 state representing the proportional coefficient that determines the proportional part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel is greater than or equal to the value of the LuxLevel Out state. Valid range: 0.0 ~ 1000.0. The default value is 80.0. The value of Light LC Regulator Kpd state

int8_t regulator_accuracy
A int8 state representing the percentage accuracy of the Light LC PI Feedback Regulator. Valid range: 0.0 ~ 100.0. The default value is 2.0. The value of Light LC Regulator Accuracy state

uint32_t set_occupancy_to_1_delay
If the message Raw field contains a Raw Value for the Time Since Motion Sensed device property, which represents a value less than or equal to the value of the Light LC Occupancy Delay state, it shall delay setting the Light LC Occupancy state to 0b1 by the difference between the value of the Light LC Occupancy Delay state and the received Time Since Motion value. The value of the difference between the value of the Light LC Occupancy Delay state and the received Time Since Motion value

struct esp_ble_mesh_light_lc_state_machine_t
Parameters of Light LC state machine

Public Members

uint8_t fade_on
The value of transition time of Light LC Time Fade On

uint8_t fade
The value of transition time of Light LC Time Fade
uint8_t fade_standby_auto
The value of transition time of Light LC Time Fade Standby Auto

uint8_t fade_standby_manual
The value of transition time of Light LC Time Fade Standby Manual

struct esp_ble_mesh_light_lc_state_machine_t::[anonymous] trans_time
The Fade On, Fade, Fade Standby Auto, and Fade Standby Manual states are transition states that define
the transition of the Lightness Out and LuxLevel Out states. This transition can be started as a result of
the Light LC State Machine change or as a result of receiving the Light LC Light OnOff Set or Light LC
Light Set Unacknowledged message. The value of transition time

esp_ble_mesh_lc_state_t state
The value of Light LC state machine state

struct k_delayed_work timer
Timer of Light LC state machine

struct esp_ble_mesh_light_control_t
Parameters of Light Lightness controller

Public Members

esp_ble_mesh_light_lc_state_t state
Parameters of Light LC state

esp_ble_mesh_light_lc_property_state_t prop_state
Parameters of Light LC Property state

esp_ble_mesh_light_lc_state_machine_t state_machine
Parameters of Light LC state machine

struct esp_ble_mesh_light_lc_srv_t
User data of Light LC Server Model

Public Members

esp_ble_mesh_model_t *model
Pointer to the Lighting LC Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl
Response control of the server model received messages

esp_ble_mesh_light_control_t *lc
Parameters of the Light controller

esp_ble_mesh_last_msg_info_t last
Parameters of the last received set message
```

**esp_ble_mesh_state_transition_t**

Parameters of state transition

**struct esp_ble_mesh_light_lc_setup_srv_t**

User data of Light LC Setup Server Model

**Public Members**

**esp_ble_mesh_model_t **

*model

Pointer to the Lighting LC Setup Server Model. Initialized internally.

**esp_ble_mesh_server_rsp_ctrl_t **

*rsp_ctrl

Response control of the server model received messages

**esp_ble_mesh_light_control_t **

*lc

Parameters of the Light controller

**struct esp_ble_mesh_state_change_light_lightness_default_set_t**

Parameter of Light Lightness Default state change event

**Public Members**

**uint16_t lightness**

The value of Light Lightness Default state

**struct esp_ble_mesh_state_change_light_lightness_range_set_t**

Parameters of Light Lightness Range state change event

**Public Members**

**uint16_t lightness**

The value of Light Lightness Range state change event
```
**Public Members**

```c
uint16_t range_min
```
The minimum value of Light Lightness Range state

```c
uint16_t range_max
```
The maximum value of Light Lightness Range state

**struct esp_ble_mesh_state_change_light_ctl_set_t**
Parameters of Light CTL state change event

**Public Members**

```c
uint16_t lightness
```
The value of Light CTL Lightness state

```c
uint16_t temperature
```
The value of Light CTL Temperature state

```c
int16_t delta_uv
```
The value of Light CTL Delta UV state

**struct esp_ble_mesh_state_change_light_ctl_temperature_set_t**
Parameters of Light CTL Temperature state change event

**Public Members**

```c
uint16_t temperature
```
The value of Light CTL Temperature state

```c
int16_t delta_uv
```
The value of Light CTL Delta UV state

**struct esp_ble_mesh_state_change_light_ctl_temperature_range_set_t**
Parameters of Light CTL Temperature Range state change event

**Public Members**

```c
uint16_t range_min
```
The minimum value of Light CTL Temperature Range state

```c
uint16_t range_max
```
The maximum value of Light CTL Temperature Range state

**struct esp_ble_mesh_state_change_light_ctl_default_set_t**
Parameters of Light CTL Default state change event
Public Members

uint16_t **lightness**
The value of Light Lightness Default state

uint16_t **temperature**
The value of Light CTL Temperature Default state

int16_t **delta_uv**
The value of Light CTL Delta UV Default state

struct esp_ble_mesh_state_change_light_hsl_set_t
Parameters of Light HSL state change event

Public Members

uint16_t **lightness**
The value of Light HSL Lightness state

uint16_t **hue**
The value of Light HSL Hue state

uint16_t **saturation**
The value of Light HSL Saturation state

struct esp_ble_mesh_state_change_light_hsl_hue_set_t
Parameter of Light HSL Hue state change event

Public Members

uint16_t **hue**
The value of Light HSL Hue state

struct esp_ble_mesh_state_change_light_hsl_saturation_set_t
Parameter of Light HSL Saturation state change event

Public Members

uint16_t **saturation**
The value of Light HSL Saturation state

struct esp_ble_mesh_state_change_light_hsl_default_set_t
Parameters of Light HSL Default state change event
Public Members

uint16_t **lightness**

The value of Light HSL Lightness Default state

uint16_t **hue**

The value of Light HSL Hue Default state

uint16_t **saturation**

The value of Light HSL Saturation Default state

struct **esp_ble_mesh_state_change_light_hsl_range_set_t**

Parameters of Light HSL Range state change event

Public Members

uint16_t **hue_range_min**

The minimum hue value of Light HSL Range state

uint16_t **hue_range_max**

The maximum hue value of Light HSL Range state

uint16_t **saturation_range_min**

The minimum saturation value of Light HSL Range state

uint16_t **saturation_range_max**

The maximum saturation value of Light HSL Range state

struct **esp_ble_mesh_state_change_light_xyl_set_t**

Parameters of Light xyl state change event

Public Members

uint16_t **lightness**

The value of Light xyl Lightness state

uint16_t **x**

The value of Light xyl x state

uint16_t **y**

The value of Light xyl y state

struct **esp_ble_mesh_state_change_light_xyl_default_set_t**

Parameters of Light xyl Default state change event
Public Members

```c
uint16_t lightness
The value of Light Lightness Default state
```

```c
uint16_t x
The value of Light xyL x Default state
```

```c
uint16_t y
The value of Light xyL y Default state
```

```c
struct esp_ble_mesh_state_change_light_xy_l_range_set_t
Parameters of Light xyL Range state change event
```

Public Members

```c
uint16_t x_range_min
The minimum value of Light xyL x Range state
```

```c
uint16_t x_range_max
The maximum value of Light xyL x Range state
```

```c
uint16_t y_range_min
The minimum value of Light xyL y Range state
```

```c
uint16_t y_range_max
The maximum value of Light xyL y Range state
```

```c
struct esp_ble_mesh_state_change_light_xy_l_range_set_t
Parameter of Light LC Mode state change event
```

Public Members

```c
uint8_t mode
The value of Light LC Mode state
```

```c
struct esp_ble_mesh_state_change_light_lc_mode_set_t
Parameter of Light LC Occupancy Mode state change event
```

Public Members

```c
uint8_t mode
The value of Light LC Occupancy Mode state
```

```c
struct esp_ble_mesh_state_change_light_lc_light_onoff_set_t
Parameter of Light LC Light OnOff state change event
```
Chapter 2. API Reference

Public Members

uint8_t onoff

The value of Light LC Light OnOff state

struct esp_ble_mesh_state_change_light_lc_property_set_t

Parameters of Light LC Property state change event

Public Members

uint16_t property_id

The property id of Light LC Property state

struct net_buf_simple *property_value

The property value of Light LC Property state

struct esp_ble_mesh_state_change_sensor_status_t

Parameters of Sensor Status state change event

Public Members

uint16_t property_id

The value of Sensor Property ID

uint8_t occupancy

The value of Light LC Occupancy state

uint32_t set_occupancy_to_1_delay

The value of Light LC Set Occupancy to 1 Delay state

uint32_t ambient_luxlevel

The value of Light LC Ambient Luxlevel state

union esp_ble_mesh_state_change_sensor_status_t::[anonymous] state

Parameters of Sensor Status related state

struct esp_ble_mesh_server_recv_light_lc_property_get_t

Context of the received Light LC Property Get message

Public Members

uint16_t property_id

Property ID identifying a Light LC Property

struct esp_ble_mesh_server_recv_light_lightness_set_t

Context of the received Light Lightness Set message
Public Members

bool op_en
    Indicate if optional parameters are included

uint16_t lightness
    Target value of light lightness actual state

uint8_t tid
    Transaction ID

uint8_t trans_time
    Time to complete state transition (optional)

uint8_t delay
    Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_lightness_linear_set_t
    Context of the received Light Lightness Linear Set message

Public Members

bool op_en
    Indicate if optional parameters are included

uint16_t lightness
    Target value of light lightness linear state

uint8_t tid
    Transaction ID

uint8_t trans_time
    Time to complete state transition (optional)

uint8_t delay
    Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_lightness_default_set_t
    Context of the received Light Lightness Default Set message

Public Members

uint16_t lightness
    The value of the Light Lightness Default state

struct esp_ble_mesh_server_recv_light_lightness_range_set_t
    Context of the received Light Lightness Range Set message
Chapter 2. API

Public Members

```c
uint16_t range_min
    Value of range min field of light lightness range state
```

```c
uint16_t range_max
    Value of range max field of light lightness range state
```

```c
struct esp_ble_mesh_server_recv_light_ctl_set_t
    Context of the received Light CTL Set message
```

Public Members

```c
bool op_en
    Indicate if optional parameters are included
```

```c
uint16_t lightness
    Target value of lightctl lightness state
```

```c
uint16_t temperature
    Target value of lightctl temperature state
```

```c
int16_t delta_uv
    Target value of lightctl delta UV state
```

```c
uint8_t tid
    Transaction ID
```

```c
uint8_t trans_time
    Time to complete state transition (optional)
```

```c
uint8_t delay
    Indicate message execution delay (C.1)
```

```c
struct esp_ble_mesh_server_recv_light_ctl_temperature_set_t
    Context of the received Light CTL Temperature Set message
```

Public Members

```c
bool op_en
    Indicate if optional parameters are included
```

```c
uint16_t temperature
    Target value of lightctl temperature state
```

```c
int16_t delta_uv
    Target value of lightctl delta UV state
```
uint8_t \texttt{tid}
    Transaction ID

uint8_t \texttt{trans\_time}
    Time to complete state transition (optional)

uint8_t \texttt{delay}
    Indicate message execution delay (C.1)

\textbf{struct \texttt{esp\_ble\_mesh\_server\_recv\_light\_ctl\_temperature\_range\_set\_t}}
    Context of the received Light CTL Temperature Range Set message

\textbf{Public Members}

uint16_t \texttt{range\_min}
    Value of temperature range min field of light ctl temperature range state

uint16_t \texttt{range\_max}
    Value of temperature range max field of light ctl temperature range state

\textbf{struct \texttt{esp\_ble\_mesh\_server\_recv\_light\_ctl\_default\_set\_t}}
    Context of the received Light CTL Default Set message

\textbf{Public Members}

uint16_t \texttt{lightness}
    Value of light lightness default state

uint16_t \texttt{temperature}
    Value of light temperature default state

int16_t \texttt{delta\_uv}
    Value of light delta UV default state

\textbf{struct \texttt{esp\_ble\_mesh\_server\_recv\_light\_hsl\_set\_t}}
    Context of the received Light HSL Set message

\textbf{Public Members}

bool \texttt{op\_en}
    Indicate if optional parameters are included

uint16_t \texttt{lightness}
    Target value of light hsl lightness state
Chapter 2. API

```c
uint16_t hue
    Target value of light hsl hue state

uint16_t saturation
    Target value of light hsl saturation state

uint8_t tid
    Transaction ID

uint8_t trans_time
    Time to complete state transition (optional)

uint8_t delay
    Indicate message execution delay (C.1)
```

```c
struct esp_ble_mesh_server_recv_light_hsl_hue_set_t
    Context of the received Light HSL Hue Set message

    Public Members

    bool op_en
        Indicate if optional parameters are included

    uint16_t hue
        Target value of light hsl hue state

    uint8_t tid
        Transaction ID

    uint8_t trans_time
        Time to complete state transition (optional)

    uint8_t delay
        Indicate message execution delay (C.1)
```

```c
struct esp_ble_mesh_server_recv_light_hsl_saturation_set_t
    Context of the received Light HSL Saturation Set message

    Public Members

    bool op_en
        Indicate if optional parameters are included

    uint16_t saturation
        Target value of light hsl hue state
```
uint8_t tid
   Transaction ID

uint8_t trans_time
   Time to complete state transition (optional)

uint8_t delay
   Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_hsl_default_set_t
   Context of the received Light HSL Default Set message

Public Members

uint16_t lightness
   Value of light lightness default state

uint16_t hue
   Value of light hue default state

uint16_t saturation
   Value of light saturation default state

struct esp_ble_mesh_server_recv_light_hsl_range_set_t
   Context of the received Light HSL Range Set message

Public Members

uint16_t hue_range_min
   Value of hue range min field of light hsl hue range state

uint16_t hue_range_max
   Value of hue range max field of light hsl hue range state

uint16_t saturation_range_min
   Value of saturation range min field of light hsl saturation range state

uint16_t saturation_range_max
   Value of saturation range max field of light hsl saturation range state

struct esp_ble_mesh_server_recv_light_xyl_set_t
   Context of the received Light xyL Set message

Public Members
bool \texttt{op\_en}

Indicate whether optional parameters included

\texttt{uint16\_t lightness}

The target value of the Light xyL Lightness state

\texttt{uint16\_t x}

The target value of the Light xyL x state

\texttt{uint16\_t y}

The target value of the Light xyL y state

\texttt{uint8\_t tid}

Transaction Identifier

\texttt{uint8\_t trans\_time}

Time to complete state transition (optional)

\texttt{uint8\_t delay}

Indicate message execution delay (C.1)

\textbf{struct esp\_ble\_mesh\_server\_recv\_light\_xyl\_default\_set\_t}

Context of the received Light xyL Default Set message

\textbf{Public Members}

\texttt{uint16\_t lightness}

The value of the Light Lightness Default state

\texttt{uint16\_t x}

The value of the Light xyL x Default state

\texttt{uint16\_t y}

The value of the Light xyL y Default state

\textbf{struct esp\_ble\_mesh\_server\_recv\_light\_xyl\_range\_set\_t}

Context of the received Light xyL Range Set message

\textbf{Public Members}

\texttt{uint16\_t x\_range\_min}

The value of the xyL x Range Min field of the Light xyL x Range state

\texttt{uint16\_t x\_range\_max}

The value of the xyL x Range Max field of the Light xyL x Range state
Chapter 2. API

uint16_t y_range_min
The value of the xyL y Range Min field of the Light xyL y Range state

uint16_t y_range_max
The value of the xyL y Range Max field of the Light xyL y Range state

struct esp_ble_mesh_server_recv_light_lc_mode_set_t
Context of the received Light LC Mode Set message

Public Members

uint8_t mode
The target value of the Light LC Mode state

struct esp_ble_mesh_server_recv_light_lc_om_set_t
Context of the received Light OM Set message

Public Members

uint8_t mode
The target value of the Light LC Occupancy Mode state

struct esp_ble_mesh_server_recv_light_lc_light_onoff_set_t
Context of the received Light LC Light OnOff Set message

Public Members

bool op_en
Indicate whether optional parameters included

uint8_t light_onoff
The target value of the Light LC Light OnOff state

uint8_t tid
Transaction Identifier

uint8_t trans_time
Time to complete state transition (optional)

uint8_t delay
Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_lc_property_set_t
Context of the received Light LC Property Set message
**Public Members**

`uint16_t property_id`
Property ID identifying a Light LC Property

`struct net_buf_simple *property_value`
Raw value for the Light LC Property

`struct esp_ble_mesh_server_recv_sensor_status_t`
Context of the received Sensor Status message

**Public Members**

`struct net_buf_simple *data`
Value of sensor data state (optional)

`struct esp_ble_mesh_lighting_server_cb_param_t`
Lighting Server Model callback parameters

**Public Members**

`esp_ble_mesh_model_t *model`
Pointer to Lighting Server Models

`esp_ble_mesh_msg_ctx_t ctx`
Context of the received messages

`esp_ble_mesh_lighting_server_cb_value_t value`
Value of the received Lighting Messages

**Macros**

`ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_CLI(cli_pub, cli_data)`
Define a new Light Lightness Client Model.

备注: This API needs to be called for each element on which the application needs to have a Light Lightness Client Model.

参数
- `cli_pub` – Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `cli_data` – Pointer to the unique struct `esp_ble_mesh_client_t`.

返回
- New Light Lightness Client Model instance.

`ESP_BLE_MESH_MODEL_LIGHT_CTL_CLI(cli_pub, cli_data)`
Define a new Light CTL Client Model.

备注: This API needs to be called for each element on which the application needs to have a Light CTL Client Model.
Chapter 2. API 参考

### 参数
- **cli_pub** — Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **cli_data** — Pointer to the unique struct `esp_ble_mesh_client_t`.

**ESP_BLE_MESH_MODEL_LIGHT_HSL_CLI** (cli_pub, cli_data)
Define a new Light HSL Client Model.

**备注**：This API needs to be called for each element on which the application needs to have a Light HSL Client Model.

### 参数
- **cli_pub** — Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **cli_data** — Pointer to the unique struct `esp_ble_mesh_client_t`.

**ESP_BLE_MESH_MODEL_LIGHT_XYL_CLI** (cli_pub, cli_data)
Define a new Light xyL Client Model.

**备注**：This API needs to be called for each element on which the application needs to have a Light xyL Client Model.

### 参数
- **cli_pub** — Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **cli_data** — Pointer to the unique struct `esp_ble_mesh_client_t`.

**ESP_BLE_MESH_MODEL_LIGHT_LC_CLI** (cli_pub, cli_data)
Define a new Light LC Client Model.

**备注**：This API needs to be called for each element on which the application needs to have a Light LC Client Model.

### 参数
- **cli_pub** — Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **cli_data** — Pointer to the unique struct `esp_ble_mesh_client_t`.

**ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_SRV** (srv_pub, srv_data)
Lighting Server Models related context.
Define a new Light Lightness Server Model.

**备注**：1. The Light Lightness Server model extends the Generic Power OnOff Server model and the Generic Level Server model. When this model is present on an Element, the corresponding Light Lightness Setup Server model shall also be present.
   a. This model shall support model publication and model subscription.

### 参数
- **srv_pub** — Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- **srv_data** — Pointer to the unique struct `esp_ble_mesh_light_lightness_srv_t`.
**Chapter 2. API Reference**

### Define a new Light Lightness Setup Server Model

#### ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_SETUP_SRV (srv_pub, srv_data)

Define a new Light Lightness Setup Server Model.

**备注:** 1. The Light Lightness Setup Server model extends the Light Lightness Server model and the Generic Power OnOff Setup Server model.

   a. This model shall support model subscription.

### Define a new Light CTL Server Model

#### ESP_BLE_MESH_MODEL_LIGHT_CTL_SRV (srv_pub, srv_data)

Define a new Light CTL Server Model.

**备注:** 1. The Light CTL Server model extends the Light Lightness Server model. When this model is present on an Element, the corresponding Light CTL Temperature Server model and the corresponding Light CTL Setup Server model shall also be present.

   a. This model shall support model publication and model subscription.

   b. The model requires two elements: the main element and the Temperature element. The Temperature element contains the corresponding Light CTL Temperature Server model and an instance of a Generic Level state bound to the Light CTL Temperature state on the Temperature element. The Light CTL Temperature state on the Temperature element is bound to the Light CTL state on the main element.

### Define a new Light CTL Setup Server Model

#### ESP_BLE_MESH_MODEL_LIGHT_CTL_SETUP_SRV (srv_pub, srv_data)

Define a new Light CTL Setup Server Model.

**备注:** 1. The Light CTL Setup Server model extends the Light CTL Server and the Light Lightness Setup Server.

   a. This model shall support model subscription.

### Define a new Light CTL Temperature Server Model

#### ESP_BLE_MESH_MODEL_LIGHT_CTL_TEMP_SRV (srv_pub, srv_data)

Define a new Light CTL Temperature Server Model.

**备注:** 1. The Light CTL Temperature Server model extends the Generic Level Server model.

   a. This model shall support model publication and model subscription.
**Chapter 2. API 参考**

**参数**
- `srv_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_light_ctl_temp_srv_t`.

返回 New Light CTL Temperature Server Model instance.

**ESP_BLE_MESH_MODEL_LIGHT_HSL_SRV** `(srv_pub, srv_data)`
Define a new Light HSL Server Model.

备注：1. The Light HSL Server model extends the Light Lightness Server model. When this model is present on an Element, the corresponding Light HSL Hue Server model and the corresponding Light HSL Saturation Server model and the corresponding Light HSL Setup Server model shall also be present.

a. This model shall support model publication and model subscription.

b. The model requires three elements: the main element and the Hue element and the Saturation element. The Hue element contains the corresponding Light HSL Hue Server model and an instance of a Generic Level state bound to the Light HSL Hue state on the Hue element. The Saturation element contains the corresponding Light HSL Saturation Server model and an instance of a Generic Level state bound to the Light HSL Saturation state on the Saturation element. The Light HSL Hue state on the Hue element is bound to the Light HSL state on the main element and the Light HSL Saturation state on the Saturation element is bound to the Light HSL state on the main element.

**参数**
- `srv_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_light_hsl_srv_t`.

返回 New Light HSL Server Model instance.

**ESP_BLE_MESH_MODEL_LIGHT_HSL_SETUP_SRV** `(srv_pub, srv_data)`
Define a new Light HSL Setup Server Model.

备注：1. The Light HSL Setup Server model extends the Light HSL Server and the Light Lightness Setup Server.

a. This model shall support model subscription.

**参数**
- `srv_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_light_hsl_setup_srv_t`.

返回 New Light HSL Setup Server Model instance.

**ESP_BLE_MESH_MODEL_LIGHT_HSL_HUE_SRV** `(srv_pub, srv_data)`
Define a new Light HSL Hue Server Model.

备注：1. The Light HSL Hue Server model extends the Generic Level Server model. This model is associated with the Light HSL Server model.

a. This model shall support model publication and model subscription.

**参数**
- `srv_pub` - Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data` - Pointer to the unique struct `esp_ble_mesh_light_hsl_hue_srv_t`.

返回 New Light HSL Hue Server Model instance.
**ESP_BLE_MESH_MODEL_LIGHT_HSL_SAT_SRV** \((\text{srv}_\text{pub}, \text{srv}_\text{data})\)

Define a new Light HSL Saturation Server Model.

**备注:** 1. The Light HSL Saturation Server model extends the Generic Level Server model. This model is associated with the Light HSL Server model.

   a. This model shall support model publication and model subscription.

   **参数**
   
   • \texttt{srv\_pub} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_model\_pub\_t}.
   • \texttt{srv\_data} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t}.

   **返回** New Light HSL Saturation Server Model instance.

**ESP_BLE_MESH_MODEL_LIGHT_XYL_SRV** \((\text{srv}_\text{pub}, \text{srv}_\text{data})\)

Define a new Light xyL Server Model.

**备注:** 1. The Light xyL Server model extends the Light Lightness Server model. When this model is present on an Element, the corresponding Light xyL Setup Server model shall also be present.

   a. This model shall support model publication and model subscription.

   **参数**
   
   • \texttt{srv\_pub} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_model\_pub\_t}.
   • \texttt{srv\_data} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_light\_xy\_srv\_t}.

   **返回** New Light xyL Server Model instance.

**ESP_BLE_MESH_MODEL_LIGHT_XYL_SETUP_SRV** \((\text{srv}_\text{pub}, \text{srv}_\text{data})\)

Define a new Light xyL Setup Server Model.

**备注:** 1. The Light xyL Setup Server model extends the Light xyL Server and the Light Lightness Setup Server.

   a. This model shall support model subscription.

   **参数**
   
   • \texttt{srv\_pub} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_model\_pub\_t}.
   • \texttt{srv\_data} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_light\_xy\_srv\_t}.

   **返回** New Light xyL Setup Server Model instance.

**ESP_BLE_MESH_MODEL_LIGHT_LC_SRV** \((\text{srv}_\text{pub}, \text{srv}_\text{data})\)

Define a new Light LC Server Model.

**备注:** 1. The Light LC (Lightness Control) Server model extends the Light Lightness Server model and the Generic OnOff Server model. When this model is present on an Element, the corresponding Light LC Setup Server model shall also be present.

   a. This model shall support model publication and model subscription.
   
   b. This model may be used to represent an element that is a client to a Sensor Server model and controls the Light Lightness Actual state via defined state bindings.

   **参数**
   
   • \texttt{srv\_pub} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_model\_pub\_t}.
   • \texttt{srv\_data} – Pointer to the unique struct \texttt{esp\_ble\_mesh\_light\_lc\_srv\_t}. 
Define a new Light LC Setup Server Model.

**Esp BLE Mesh Model Light LC Setup_SRV**(srv_pub, srv_data)

New Light LC Server Model instance.

**Type Definitions**

```c
typedef void (*esp_ble_mesh_light_client_cb_t)(esp_ble_mesh_light_client_cb_event_t event,
 esp_ble_mesh_light_client_cb_param_t *param)
```

Bluetooth Mesh Light Client Model function.

**Lighting Client Model callback function type**

- **Param event** Event type
- **Param param** Pointer to callback parameter

```c
typedef void (*esp_ble_mesh_lighting_server_cb_t)(esp_ble_mesh_lighting_server_cb_event_t event,
 esp_ble_mesh_lighting_server_cb_param_t *param)
```

Bluetooth Mesh Lighting Server Model function.

**Lighting Server Model callback function type**

- **Param event** Event type
- **Param param** Pointer to callback parameter

**Enumerations**

```c
typedef enum esp_ble_mesh_light_client_cb_event_t

This enum value is the event of Lighting Client Model

**Values:**

- **enumerator ESP_BLE_MESH_LIGHT_CLIENT_GET_STATE_EVT**
- **enumerator ESP_BLE_MESH_LIGHT_CLIENT_SET_STATE_EVT**
- **enumerator ESP_BLE_MESH_LIGHT_CLIENT_PUBLISH_EVT**
- **enumerator ESP_BLE_MESH_LIGHT_CLIENT_TIMEOUT_EVT**

```c
typedef enum esp_ble_mesh_lc_state_t

This enum value is the Light LC State Machine states

**Values:**

- **enumerator ESP_BLE_MESH_LIGHT_CLIENT_EVT_MAX**
```
Chapter 2. API

enumerator `ESP_BLE_MESH_LC_OFF`

enumerator `ESP_BLE_MESH_LC_STANDBY`

enumerator `ESP_BLE_MESH_LC_FADE_ON`

enumerator `ESP_BLE_MESH_LC_RUN`

enumerator `ESP_BLE_MESH_LC_FADE`

enumerator `ESP_BLE_MESH_LC_PROLONG`

enumerator `ESP_BLE_MESH_LC_FADE_STANDBY_AUTO`

enumerator `ESP_BLE_MESH_LC_FADE_STANDBY_MANUAL`

enum `esp_ble_mesh_lighting_server_cb_event_t`

This enum value is the event of Lighting Server Model

Values:

enumerator `ESP_BLE_MESH_LIGHTING_SERVER_STATE_CHANGE_EVT`

i. When `get_auto_rsp` is set to `ESP_BLE_MESH_SERVER_AUTO_RSP`, no event will be callback to the application layer when Lighting Get messages are received.

ii. When `set_auto_rsp` is set to `ESP_BLE_MESH_SERVER_AUTO_RSP`, this event will be callback to the application layer when Lighting Set/Set Unack messages are received.

enumerator `ESP_BLE_MESH_LIGHTING_SERVER_RECV_GET_MSG_EVT`

When `get_auto_rsp` is set to `ESP_BLE_MESH_SERVER_RSP_BY_APP`, this event will be callback to the application layer when Lighting Get messages are received.

enumerator `ESP_BLE_MESH_LIGHTING_SERVER_RECV_SET_MSG_EVT`

When `set_auto_rsp` is set to `ESP_BLE_MESH_SERVER_RSP_BY_APP`, this event will be callback to the application layer when Lighting Set/Set Unack messages are received.

enumerator `ESP_BLE_MESH_LIGHTING_SERVER_RECV_STATUS_MSG_EVT`

When `status_auto_rsp` is set to `ESP_BLE_MESH_SERVER_RSP_BY_APP`, this event will be callback to the application layer when Sensor Status message is received.

enumerator `ESP_BLE_MESH_LIGHTING_SERVER_EVT_MAX`

2.3.6 NimBLE-based host APIs

Overview

Apache MyNewt NimBLE is a highly configurable and BT SIG qualifiable BLE stack providing both host and controller functionalities. ESP-IDF supports NimBLE host stack which is specifically ported for ESP32 platform and FreeRTOS. The underlying controller is still the same (as in case of Bluedroid) providing VHCI interface. Refer to NimBLE user guide for a complete list of features and additional information on NimBLE stack. Most features
of NimBLE including BLE Mesh are supported by ESP-IDF. The porting layer is kept cleaner by maintaining all
the existing APIs of NimBLE along with a single ESP-NimBLE API for initialization, making it simpler for the
application developers.

Architecture

Currently, NimBLE host and controller support different transports such as UART and RAM between them. How-
ever, RAM transport cannot be used as is in case of ESP as ESP controller supports VHCI interface and buffering
schemes used by NimBLE host is incompatible with that used by ESP controller. Therefore, a new transport between
NimBLE host and ESP controller has been added. This is depicted in the figure below. This layer is responsible for
maintaining pool of transport buffers and formatting buffers exchanges between host and controller as per the re-
quirements.

![ESP NimBLE Stack](image)

Thrading Model

The NimBLE host can run inside the application thread or can have its own independent thread. This flex-
ibility is inherently provided by NimBLE design. By default, a thread is spawned by the porting function
nimble_port_freertos_init. This behavior can be changed by overriding the same function. For BLE
Mesh, additional thread (advertising thread) is used which keeps on feeding advertisement events to the main thread.

Programming Sequence

To begin with, make sure that the NimBLE stack is enabled from menuconfig choose NimBLE for the Bluetooth host.

Typical programming sequence with NimBLE stack consists of the following steps:

- Initialize NVS flash using nvs_flash_init() API. This is because ESP controller uses NVS during
  initialization.
- Initialize the host and controller stack using nimble_port_init.
- Initialize the required NimBLE host configuration parameters and callbacks
- Perform application specific tasks_INITIALIZATION
- Run the thread for host stack using nimble_port_freertos_init

This documentation does not cover NimBLE APIs. Refer to NimBLE tutorial for more details on the programming
sequence/NimBLE APIs for different scenarios.

API Reference

Header File

- components/bt/host/nimble/esp-hci/include/esp_nimble_hci.h

Functions
Chapter 2. API

**esp_err_t esp_nimble_hci_init**(void)

Initialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

This function initializes the transport buffers to be exchanged between NimBLE host and ESP controller. It also registers required host callbacks with the controller.

**返回**
- ESP_OK if the initialization is successful
- Appropriate error code from esp_err_t in case of an error

**esp_err_t esp_nimble_hci_deinit**(void)

Deinitialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

**备注**: This function should be called after the NimBLE host is deinitialized.

**返回**
- ESP_OK if the deinitialization is successful
- Appropriate error codes from esp_err_t in case of an error

**Macros**

BLE_HCI_UART_H4_NONE

BLE_HCI_UART_H4_CMD

BLE_HCI_UART_H4_ACL

BLE_HCI_UART_H4_SCO

BLE_HCI_UART_H4_EVT

ESP-IDF 目前支持两个主机堆栈，基于 Bluedroid 的堆栈（默认）支持传统蓝牙和 BLE，而基于 Apache NimBLE 的堆栈仅支持 BLE，用户可参考如下信息进行选择:

- 对于同时涉及传统蓝牙和 BLE 的用例，应该选用 Bluedroid。
- 对于仅涉及 BLE 的用例，建议选用 NimBLE。在代码占用和运行时，NimBLE 对内存的要求较低，因此适用于此类场景。

请点击下方链接，查看 ESP32 蓝牙架构:

- [ESP32 蓝牙架构 (PDF)]

蓝牙 API 的示例代码存放于 ESP-IDF 示例项目的 bluetooth/bluedroid 目录下。

下面的示例给出了详细介绍:

- GATT 客户端示例
- GATT 服务器服务表格示例
- GATT 服务器端示例
- GATT 客户端安全性示例
- GATT 服务器安全性示例
- GATT 客户端多连接示例

### 2.4 Error Codes Reference

This section lists various error code constants defined in ESP-IDF.

For general information about error codes in ESP-IDF, see *Error Handling*. 

Espressif Systems 778 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
Chapter 2. API

ESP_FAIL (-1): Generic esp_err_t code indicating failure
ESP_OK (0): esp_err_t value indicating success (no error)
ESP_ERR_NO_MEM (0x101): Out of memory
ESP_ERR_INVALID_ARG (0x102): Invalid argument
ESP_ERR_INVALID_STATE (0x103): Invalid state
ESP_ERR_INVALID_SIZE (0x104): Invalid size
ESP_ERR_NOT_FOUND (0x105): Requested resource not found
ESP_ERR_NOT_SUPPORTED (0x106): Operation or feature not supported
ESP_ERR_TIMEOUT (0x107): Operation timed out
ESP_ERR_INVALID_RESPONSE (0x108): Received response was invalid
ESP_ERR_INVALID_CRC (0x109): CRC or checksum was invalid
ESP_ERR_INVALID_VERSION (0x10a): Version was invalid
ESP_ERR_INVALID_MAC (0x10b): MAC address was invalid
ESP_ERR_NOT_FINISHED (0x10c): There are items remained to retrieve
ESP_ERR_NVS_BASE (0x1100): Starting number of error codes
ESP_ERR_NVS_NOT_INITIALIZED (0x1101): The storage driver is not initialized
ESP_ERR_NVS_NOT_FOUND (0x1102): A requested entry couldn’t be found or namespace doesn’t exist yet and mode is NVS_READONLY
ESP_ERR_NVS_TYPE_MISMATCH (0x1103): The type of set or get operation doesn’t match the type of value stored in NVS
ESP_ERR_NVS_READ_ONLY (0x1104): Storage handle was opened as read only
ESP_ERR_NVS_NOT_ENOUGH_SPACE (0x1105): There is not enough space in the underlying storage to save the value
ESP_ERR_NVS_INVALID_NAME (0x1106): Namespace name doesn’t satisfy constraints
ESP_ERR_NVS_INVALID_HANDLE (0x1107): Handle has been closed or is NULL
ESP_ERR_NVS_REMOVE_FAILED (0x1108): The value wasn’t updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn’t fail again.
ESP_ERR_NVS_KEY_TOO_LONG (0x1109): Key name is too long
ESP_ERR_NVS_PAGE_FULL (0x110a): Internal error; never returned by nvs API functions
ESP_ERR_NVS_INVALID_STATE (0x110b): NVS is in an inconsistent state due to a previous error. Call nvs_flash_init and nvs_open again, then retry.
ESP_ERR_NVS_INVALID_LENGTH (0x110c): String or blob length is not sufficient to store data
ESP_ERR_NVS_NO_FREE_PAGES (0x110d): NVS partition doesn’t contain any empty pages. This may happen if NVS partition was truncated. Erase the whole partition and call nvs_flash_init again.
ESP_ERR_NVS_VALUE_TOO_LONG (0x110e): Value doesn’t fit into the entry or string or blob length is longer than supported by the implementation
ESP_ERR_NVS_PART_NOT_FOUND (0x110f): Partition with specified name is not found in the partition table
ESP_ERR_NVS_NEW_VERSION_FOUND (0x1110): NVS partition contains data in new format and cannot be recognized by this version of code
ESP_ERR_NVS_XTS_ENCR_FAILED (0x1111): XTS encryption failed while writing NVS entry
ESP_ERR_NVS_XTS_DECR_FAILED (0x1112): XTS decryption failed while reading NVS entry
**ESP_ERR_NVS_XTS_CFG_FAILED** (0x1113): XTS configuration setting failed

**ESP_ERR_NVS_XTS_CFG_NOT_FOUND** (0x1114): XTS configuration not found

**ESP_ERR_NVS_ENCR_NOT_SUPPORTED** (0x1115): NVS encryption is not supported in this version

**ESP_ERR_NVS_KEYS_NOT_INITIALIZED** (0x1116): NVS key partition is uninitialized

**ESP_ERR_NVS_CORRUPT_KEY_PART** (0x1117): NVS key partition is corrupt

**ESP_ERR_NVS_CONTENT_DIFFERS** (0x1118): Internal error; never returned by nvs API functions. NVS key is different in comparison

**ESP_ERR_NVS_WRONG_ENCRYPTION** (0x1119): NVS partition is marked as encrypted with generic flash encryption. This is forbidden since the NVS encryption works differently.

**ESP_ERR_ULP_BASE** (0x1200): Offset for ULP-related error codes

**ESP_ERR_ULP_SIZE_TOO_BIG** (0x1201): Program doesn’t fit into RTC memory reserved for the ULP

**ESP_ERR_ULP_INVALID_LOAD_ADDR** (0x1202): Load address is outside of RTC memory reserved for the ULP

**ESP_ERR_ULP_DUPLICATE_LABEL** (0x1203): More than one label with the same number was defined

**ESP_ERR_ULP_UNDEFINED_LABEL** (0x1204): Branch instructions references an undefined label

**ESP_ERR_ULP_BRANCH_OUT_OF_RANGE** (0x1205): Branch target is out of range of B instruction (try replacing with BX)

**ESP_ERR_OTA_BASE** (0x1500): Base error code for ota_ops api

**ESP_ERR_OTA_PARTITION_CONFLICT** (0x1501): Error if request was to write or erase the current running partition

**ESP_ERR_OTA_SELECT_INFO_INVALID** (0x1502): Error if OTA data partition contains invalid content

**ESP_ERR_OTA_VALIDATE_FAILED** (0x1503): Error if OTA app image is invalid

**ESP_ERR_OTA_SMALL_SEC_VER** (0x1504): Error if the firmware has a secure version less than the running firmware.

**ESP_ERR_OTA_ROLLBACK_FAILED** (0x1505): Error if flash does not have valid firmware in passive partition and hence rollback is not possible

**ESP_ERR_OTA_ROLLBACK_INVALID_STATE** (0x1506): Error if current active firmware is still marked in pending validation state (ESP_OTA_IMG_PENDING_VERIFY), essentially first boot of firmware image post upgrade and hence firmware upgrade is not possible

**ESP_ERR_EFUSE** (0x1600): Base error code for efuse api.

**ESP_OK_EFUSE_CNT** (0x1601): OK the required number of bits is set.

**ESP_ERR_EFUSE_CNT_IS_FULL** (0x1602): Error field is full.

**ESP_ERR_EFUSE_REPEATED_PROG** (0x1603): Error repeated programming of programmed bits is strictly forbidden.

**ESP_ERR_CODING** (0x1604): Error while a encoding operation.

**ESP_ERR_NOT_ENOUGH_UNUSED_KEY_BLOCKS** (0x1605): Error not enough unused key blocks available

**ESP_ERR_DAMAGED_READING** (0x1606): Error. Burn or reset was done during a reading operation leads to damage read data. This error is internal to the efuse component and not returned by any public API.

**ESP_ERR_IMAGE_BASE** (0x2000)

**ESP_ERR_IMAGE_FLASH_FAIL** (0x2001)

**ESP_ERR_IMAGE_INVALID** (0x2002)

**ESP_ERR_WIFI_BASE** (0x3000): Starting number of WiFi error codes

**ESP_ERR_WIFI_NOT_INIT** (0x3001): WiFi driver was not installed by esp_wifi_init
ESP_ERR_WIFI_NOT_STARTED (0x3002): WiFi driver was not started by esp_wifi_start
ESP_ERR_WIFI_NOT_STOPPED (0x3003): WiFi driver was not stopped by esp_wifi_stop
ESP_ERR_WIFI_IF (0x3004): WiFi interface error
ESP_ERR_WIFI_MODE (0x3005): WiFi mode error
ESP_ERR_WIFI_STATE (0x3006): WiFi internal state error
ESP_ERR_WIFI_CONN (0x3007): WiFi internal control block of station or soft-AP error
ESP_ERR_WIFI_NV (0x3008): WiFi internal NVS module error
ESP_ERR_WIFI_MAC (0x3009): MAC address is invalid
ESP_ERR_WIFI_SSID (0x300a): SSID is invalid
ESP_ERR_WIFI_PASSWORD (0x300b): Password is invalid
ESP_ERR_WIFI_TIMEOUT (0x300c): Timeout error
ESP_ERR_WIFI_WAKE_FAIL (0x300d): WiFi is in sleep state(RF closed) and wakeup fail
ESP_ERR_WIFI_WOULD_BLOCK (0x300e): The caller would block
ESP_ERR_WIFI_NOT_CONNECT (0x300f): Station still in disconnect status
ESP_ERR_WIFI_POST (0x3012): Failed to post the event to WiFi task
ESP_ERR_WIFI_INIT_STATE (0x3013): Invalid WiFi state when init/deinit is called
ESP_ERR_WIFI_STOP_STATE (0x3014): Returned when WiFi is stopping
ESP_ERR_WIFI_NOT_ASSOC (0x3015): The WiFi connection is not associated
ESP_ERR_WIFI_TX_DISALLOW (0x3016): The WiFi TX is disallowed
ESP_ERR_WIFI_REGISTRAR (0x3033): WPS registrar is not supported
ESP_ERR_WIFI_WPS_TYPE (0x3034): WPS type error
ESP_ERR_WIFI_WPS_SM (0x3035): WPS state machine is not initialized
ESP_ERR_ESPNOW_BASE (0x3064): ESPNOW error number base.
ESP_ERR_ESPNOW_NOT_INIT (0x3065): ESPNOW is not initialized.
ESP_ERR_ESPNOW_ARG (0x3066): Invalid argument
ESP_ERR_ESPNOW_NO_MEM (0x3067): Out of memory
ESP_ERR_ESPNOW_FULL (0x3068): ESPNOW peer list is full
ESP_ERR_ESPNOW_NOT_FOUND (0x3069): ESPNOW peer is not found
ESP_ERR_ESPNOW_INTERNAL (0x306a): Internal error
ESP_ERR_ESPNOW_EXIST (0x306b): ESPNOW peer has existed
ESP_ERR_ESPNOW_IF (0x306c): Interface error
ESP_ERR_DPP_FAILURE (0x3097): Generic failure during DPP Operation
ESP_ERR_DPP_TX_FAILURE (0x3098): DPP Frame Tx failed OR not Acked
ESP_ERR_DPP_INVALID_ATTR (0x3099): Encountered invalid DPP Attribute
ESP_ERR_MESH_BASE (0x4000): Starting number of MESH error codes
ESP_ERR_MESH_WIFI_NOT_START (0x4001)
ESP_ERR_MESH_NOT_INIT (0x4002)
ESP_ERR_MESH_NOT_CONFIG (0x4003)
ESP_ERR_MESH_NOT_START (0x4004)
ESP_ERR_MESH_NOT_SUPPORT (0x4005)
ESP_ERR_MESH_NOT_ALLOWED (0x4006)
ESP_ERR_MESH_NO_MEMORY (0x4007)
ESP_ERR_MESH_ARGUMENT (0x4008)
ESP_ERR_MESH_EXCEED_MTU (0x4009)
ESP_ERR_MESH_TIMEOUT (0x400a)
ESP_ERR_MESH_DISCONNECTED (0x400b)
ESP_ERR_MESH_QUEUE_FAIL (0x400c)
ESP_ERR_MESH_QUEUE_FULL (0x400d)
ESP_ERR_MESH_NO_PARENT_FOUND (0x400e)
ESP_ERR_MESH_NO_ROUTE_FOUND (0x400f)
ESP_ERR_MESH_OPTION_NULL (0x4010)
ESP_ERR_MESH_OPTION_UNKNOWN (0x4011)
ESP_ERR_MESH_XON_NO_WINDOW (0x4012)
ESP_ERR_MESH_INTERFACE (0x4013)
ESP_ERR_MESH_DISCARD_DUPLICATE (0x4014)
ESP_ERR_MESH_DISCARD (0x4015)
ESP_ERR_MESH_VOTING (0x4016)
ESP_ERR_MESH_XMIT (0x4017)
ESP_ERR_MESH_QUEUE_READ (0x4018)
ESP_ERR_MESH_PS (0x4019)
ESP_ERR_MESH_RECV_RELEASE (0x401a)
ESP_ERR_ESP_NETIF_BASE (0x5000)
ESP_ERR_ESP_NETIF_INVALID_PARAMS (0x5001)
ESP_ERR_ESP_NETIF_IF_NOT_READY (0x5002)
ESP_ERR_ESP_NETIF_DHCPC_START_FAILED (0x5003)
ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED (0x5004)
ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED (0x5005)
ESP_ERR_ESP_NETIF_NO_MEM (0x5006)
ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED (0x5007)
ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED (0x5008)
ESP_ERR_ESP_NETIF_INIT_FAILED (0x5009)
ESP_ERR_ESP_NETIF_DNS_NOT_CONFIGURED (0x500a)
ESP_ERR_ESP_NETIF_MLD6_FAILED (0x500b)
ESP_ERR_ESP_NETIF_IP6_ADDR_FAILED (0x500c)
ESP_ERR_ESP_NETIF_DHCPS_START_FAILED (0x500d)
ESP_ERR_FLASH_BASE (0x6000): Starting number of flash error codes
ESP_ERR_FLASH_OP_FAIL (0x6001)
ESP_ERR_FLASH_OP_TIMEOUT (0x6002)
Chapter 2. API

ESP_ERR_FLASH_NOT_INITIALLIZED (0x6003)
ESP_ERR_FLASH_UNSUPPORTED_HOST (0x6004)
ESP_ERR_FLASH_UNSUPPORTED_CHIP (0x6005)
ESP_ERR_FLASH_PROTECTED (0x6006)
ESP_ERR_HTTP_BASE (0x7000): Starting number of HTTP error codes
ESP_ERR_HTTP_MAX_REDIRECT (0x7001): The error exceeds the number of HTTP redirects
ESP_ERR_HTTP_CONNECT (0x7002): Error open the HTTP connection
ESP_ERR_HTTP_WRITE_DATA (0x7003): Error write HTTP data
ESP_ERR_HTTP_FETCH_HEADER (0x7004): Error read HTTP header from server
ESP_ERR_HTTP_INVALID_TRANSPORT (0x7005): There are no transport support for the input scheme
ESP_ERR_HTTP_CONNECTING (0x7006): HTTP connection hasn’t been established yet
ESP_ERR_HTTP_EAGAIN (0x7007): Mapping of errno EAGAIN to esp_err_t
ESP_ERR_HTTP_CONNECTION_CLOSED (0x7008): Read FIN from peer and the connection closed
ESP_ERR_ESP_TLS_BASE (0x8000): Starting number of ESP-TLS error codes
ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME (0x8001): Error if hostname couldn’t be resolved upon tls connection
ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET (0x8002): Failed to create socket
ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY (0x8003): Unsupported protocol family
ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST (0x8004): Failed to connect to host
ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED (0x8005): failed to set/get socket option
ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT (0x8006): new connection in esp_tls_low_level_conn connection timeouted
ESP_ERR_ESP_TLS_SE_FAILED (0x8007)
ESP_ERR_ESP_TLS_TCP_CLOSED_FIN (0x8008)
ESP_ERR_MBEDTLS_CERT_PARTLY_OK (0x8010): mbedtls parse certificates was partly successful
ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED (0x8011): mbedtls api returned error
ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED (0x8012): mbedtls api returned error
ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED (0x8013): mbedtls api returned error
ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED (0x8014): mbedtls api returned error
ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED (0x8015): mbedtls api returned error
ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED (0x8016): mbedtls api returned error
ESP_ERR_MBEDTLS_SSL_SETUP_FAILED (0x8017): mbedtls api returned error
ESP_ERR_MBEDTLS_SSL_WRITE_FAILED (0x8018): mbedtls api returned error
ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED (0x8019): mbedtls api returned failed
ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED (0x801a): mbedtls api returned failed
ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED (0x801b): mbedtls api returned failed
ESP_ERR_MBEDTLS_SSL_TICKET_SETUP_FAILED (0x801c): mbedtls api returned failed
ESP_ERR_WOLFSSL_SSL_SET_HOSTNAME_FAILED (0x8031): wolfSSL api returned error
ESP_ERR_WOLFSSL_SSL_CONF_ALPN_PROTOCOLS_FAILED (0x8032): wolfSSL api returned error
ESP_ERR_WOLFSSL_CERT_VERIFY_SETUP_FAILED (0x8033): wolfSSL api returned error
Chapter 2. API

ESP_ERR_WOLFSSL_KEY_VERIFY_SETUP_FAILED (0x8034): wolfSSL api returned error
ESP_ERR_WOLFSSL_SSL_HANDSHAKE_FAILED (0x8035): wolfSSL api returned failed
ESP_ERR_WOLFSSL_CTX_SETUP_FAILED (0x8036): wolfSSL api returned failed
ESP_ERR_WOLFSSL_SSL_SETUP_FAILED (0x8037): wolfSSL api returned failed
ESP_ERR_WOLFSSL_SSL_WRITE_FAILED (0x8038): wolfSSL api returned failed
ESP_ERR_HTTPS_OTA_BASE (0x9000)
ESP_ERR_HTTPS_OTA_IN_PROGRESS (0x9001)
ESP_ERR_PING_BASE (0xa000)
ESP_ERR_PING_INVALID_PARAMS (0xa001)
ESP_ERR_PING_NO_MEM (0xa002)
ESP_ERR_HTTPD_BASE (0xb000): Starting number of HTTPD error codes
ESP_ERR_HTTPD_HANDLERS_FULL (0xb001): All slots for registering URI handlers have been consumed
ESP_ERR_HTTPD_HANDLER_EXISTS (0xb002): URI handler with same method and target URI already registered
ESP_ERR_HTTPD_INVALID_REQ (0xb003): Invalid request pointer
ESP_ERR_HTTPD_RESULT_TRUNC (0xb004): Result string truncated
ESP_ERR_HTTPD_RESP_HDR (0xb005): Response header field larger than supported
ESP_ERR_HTTPD_RESP_SEND (0xb006): Error occurred while sending response packet
ESP_ERR_HTTPD_ALLOC_MEM (0xb007): Failed to dynamically allocate memory for resource
ESP_ERR_HTTPD_TASK (0xb008): Failed to launch server task/thread
ESP_ERR_HW_CRYPTO_BASE (0xc000): Starting number of HW cryptography module error codes
ESP_ERR_HW_CRYPTO_DS_HMAC_FAIL (0xc001): HMAC peripheral problem
ESP_ERR_HW_CRYPTO_DS_INVALID_KEY (0xc002)
ESP_ERR_HW_CRYPTO_DS_INVALID_DIGEST (0xc004)
ESP_ERR_HW_CRYPTO_DS_INVALID_PADDING (0xc005)
ESP_ERR_MEMPROT_BASE (0xd000): Starting number of Memory Protection API error codes
ESP_ERR_MEMPROT_MEMORY_TYPE_INVALID (0xd001)
ESP_ERR_MEMPROT_SPLIT_ADDR_INVALID (0xd002)
ESP_ERR_MEMPROT_SPLIT_ADDR_OUT_OF_RANGE (0xd003)
ESP_ERR_MEMPROT_SPLIT_ADDR_UNALIGNED (0xd004)
ESP_ERR_MEMPROT_UNIMGMT_BLOCK_INVALID (0xd005)
ESP_ERR_MEMPROT_WORLD_INVALID (0xd006)
ESP_ERR_MEMPROT_AREA_INVALID (0xd007)
ESP_ERR_MEMPROT_CPUID_INVALID (0xd008)
ESP_ERR_TCP_TRANSPORT_BASE (0xe000): Starting number of TCP Transport error codes
ESP_ERR_TCP_TRANSPORT_CONNECTION_TIMEOUT (0xe001): Connection has timed out
ESP_ERR_TCP_TRANSPORT_CONNECTION_CLOSED_BY_FIN (0xe002): Read FIN from peer and the connection has closed (in a clean way)
ESP_ERR_TCP_TRANSPORT_CONNECTION_FAILED (0xe003): Failed to connect to the peer
ESP_ERR_TCP_TRANSPORT_NO_MEM (0xe004): Memory allocation failed
2.5 连网 API

2.5.1 Wi-Fi

ESP-NOW

概述 ESP-NOW 是一种由乐鑫公司定义的无连接 Wi-Fi 通信协议。在 ESP-NOW 中，应用程序数据被封装在各个供应商的动作帧中，然后在无连接的情况下，从一个 Wi-Fi 设备传输到另一个 Wi-Fi 设备。CTR 与 CBC-MAC 协议 (CCMP) 可用来保护动作帧的安全。ESP-NOW 广泛应用于智能照明、远程控制、传感器等领域。

帧格式 ESP-NOW 使用各个供应商的动作帧传输数据，默认比特率为 1 Mbps。各个供应商的动作帧格式为:

<table>
<thead>
<tr>
<th>MAC 报头</th>
<th>分类代码</th>
<th>组织标识符</th>
<th>随机值</th>
<th>供应商特定内容</th>
<th>FCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 字节</td>
<td>1 字节</td>
<td>3 字节</td>
<td>4 字节</td>
<td>7~255 字节</td>
<td>4 字节</td>
</tr>
</tbody>
</table>

- 分类代码：分类代码字段可用于指示各个供应商的类别（比如 127）。
- 组织标识符：组织标识符包含一个唯一标识符（比如 0x18fe34），为乐鑫指定的 MAC 地址的前三个字符。
- 随机值：防止重放攻击。
- 供应商特定内容：供应商特定内容包含供应商特定字段，如下所示:

<table>
<thead>
<tr>
<th>元素 ID</th>
<th>长度</th>
<th>组织标识符</th>
<th>类型</th>
<th>版本</th>
<th>正文</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 字节</td>
<td>1 字节</td>
<td>3 字节</td>
<td>1 字节</td>
<td>1 字节</td>
<td>0~250 字节</td>
</tr>
</tbody>
</table>

- 元素 ID：元素 ID 字段可用于指示特定于供应商的元素。
- 长度：长度是组织标识符，类型、版本和正文的总长度。
- 组织标识符：组织标识符包含一个唯一标识符（比如 0x18fe34），为乐鑫指定的 MAC 地址的前三个字符。
- 类型：类型字段设置为 4，代表 ESP-NOW。
- 版本：版本字段设置为 ESP-NOW 的版本。
- 正文：正文包含 ESP-NOW 数据。

由于 ESP-NOW 是无连接的，因此 MAC 报头与标准帧略有不同。FrameControl 字段的 FromDS 和 ToDS 位均为 0。第一个地址字段用于配置目标地址。第二个地址字段用于配置源地址。第三个地址字段用于配置广播地址 (0xff:0xff:0xff:0xff:0xff:0xff)。

安全

ESP-NOW 采用 CCMP 方法保护供应商特定动作帧的安全，具体可参考 IEEE Std. 802.11-2012。Wi-Fi 设备维护一个初始 PMK（Pre-Shared Key），PMK 可使用 AES-128 算法加密 LMK。请调用 esp_now_set_pmk() 设置 PMK，如果未设置 PMK，将使用默认 PMK。
Chapter 2. API 参考

- LMK 可通过 CCMP 方法对供应商特定的动作帧进行加密，最多拥有 6 个不同的 LMK。如果未设置配置设备的 LMK，则动作帧不进行加密。

目前，不支持加密组播供应商特定的动作帧。

**初始化和反初始化** 调用 `esp_now_init()` 初始化 ESP-NOW，调用 `esp_now_deinit()` 反初始化 ESP-NOW。ESP-NOW 数据必须在 Wi-Fi 启动前传输，因此建议在初始化 ESP-NOW 之前启动 Wi-Fi，并在反初始化 ESP-NOW 之后停止 Wi-Fi。当调用 `esp_now_deinit()` 时，配对设备的所有信息都将被删除。

**添加配对设备** 在将数据发送到其他设备之前，请先调用 `esp_now_add_peer()` 将其添加到配对设备列表中。如果启用了加密，则必须设置 LMK。ESP-NOW 数据可以从 Station 或 Softap 接口发送。确保在发送 ESP-NOW 数据之前已启用该接口。

配对设备的最大数量是 20，其中加密设备的数量不超过 16，默认值是 6。如果想要修改加密设备的数量，在 WiFi menuconfig 设置 `CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM`。

在发送广播数据之前必须添加具有广播 MAC 地址的设备。配对设备的信道范围是从 0 - 14。如果信道设置为 0，数据将在当前信道上发送。否则，必须使用本地设备所在的信道。

**发送 ESP-NOW 数据** 调用 `esp_now_send()` 发送 ESP-NOW 数据。调用 `esp_now_register_send_cb()` 注册发送回调函数。如果 MAC 层成功接收到数据，则该函数将返回 `ESP_NOW_SEND_SUCCESS` 事件。否则，它将返回 `ESP_NOW_SEND_FAIL`。ESP-NOW 数据发送失败可能有几种原因，比如目标设备不存在、设备的信道不相同、动作帧在传输过程中丢失等。应用层并不一定可以接收到数据。如果需要，应用层可在接收 ESP-NOW 数据时发送一个应答 (ACK) 数据。如果接收 ACK 数据超时，则将重新传输 ESP-NOW 数据。可以为 ESP-NOW 数据设置序列号，从而删除重复的数据。

如果有大量 ESP-NOW 数据要发送，调用 `esp_now_send()` 时需注意单次发送的数据不能超过 250 字节。请注意，两个 ESP-NOW 数据包的发送间隔太短可能导致回调函数返回混乱。因此，建议在发送 ESP-NOW 数据之前等一个应答 (ACK) 数据。发送回调函数在高优先级的 Wi-Fi 任务中运行。因此，不要在回调函数中执行冗长的操作。相反，将必要的数据发送到队列，并交给优先级较低的任务处理。

**接收 ESP-NOW 数据** 调用 `esp_now_register_recv_cb()` 注册接收回调函数。当接收 ESP-NOW 数据时，需要调用接收回调函数。接收回调函数也在 Wi-Fi 任务任务中运行。因此，不要在回调函数中执行冗长的操作。相反，将必要的数据发送到队列，并交给优先级较低的任务处理。

**配置 ESP-NOW 速率** 调用 `esp_wifi_config_espnow_rate()` 配置指定接口的 ESPNOW 速率。确保在配置速率之前使能接口。这个 API 应该在 `esp_wifi_start()` 之后调用。

**配置 ESP-NOW 功耗参数** 当且仅当 ESP32 配置为 STA 模式时，允许其进行休眠。

进行休眠时，调用 `esp_now_set_wake_window()` 为 ESP-NOW 收包配置 Window。默认情况下 Window 为最大值，将允许一直收包。

如果对 ESP-NOW 进功耗管理，也需要调用 `esp_wifi_connectionless_module_set_wake_interval()`。

请参考非连接模块功耗管理 获取更多信息。

**应用示例**

- 如何在设备间传输 ESP-NOW 数据：`wifi/espnsw`。
- 了解更多 ESP-NOW 的应用示例，请参考 README.md 文件。

**API 参考**

Espressif Systems 786

Submit Document Feedback
Header File

- components/esp_wifi/include/esp_now.h

Functions

```c
esp_err_t esp_now_init (void)
Initialize ESPNOW function.
```

- ESP_OK: succeed
- ESP_ERR_ESPNOW_INTERNAL: Internal error

```c
esp_err_t esp_now_deinit (void)
De-initialize ESPNOW function.
```

- ESP_OK: succeed

```c
esp_err_t esp_now_get_version (uint32_t *version)
Get the version of ESPNOW.
```

- ESP_OK: succeed
- ESP_ERR_ESPNOW_ARG: invalid argument

```c
esp_err_t esp_now_register_recv_cb (esp_now_recv_cb_t cb)
Register callback function of receiving ESPNOW data.
```

- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
- ESP_ERR_ESPNOW_INTERNAL: internal error

```c
esp_err_t esp_now_unregister_recv_cb (void)
Unregister callback function of receiving ESPNOW data.
```

- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized

```c
esp_err_t esp_now_register_send_cb (esp_now_send_cb_t cb)
Register callback function of sending ESPNOW data.
```

- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized

```c
esp_err_t esp_now_unregister_send_cb (void)
Unregister callback function of sending ESPNOW data.
```

- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized

```c
esp_err_t esp_now_send (const uint8_t *peer_addr, const uint8_t *data, size_t len)
Send ESPNOW data.
```

Attention 1. If peer_addr is not NULL, send data to the peer whose MAC address matches peer_addr
Attention 2. If peer_addr is NULL, send data to all of the peers that are added to the peer list
Chapter 2. API

3. The maximum length of data must be less than ESP_NOW_MAX_DATA_LEN
4. The buffer pointed to by data argument does not need to be valid after esp_now_send returns

参数
- `peer_addr` – peer MAC address
- `data` – data to send
- `len` – length of data

返回
- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG: invalid argument
- ESP_ERR_ESPNOW_INTERNAL: internal error
- ESP_ERR_ESPNOW_NO_MEM: out of memory, when this happens, you can delay a while before sending the next data
- ESP_ERR_ESPNOW_NOT_FOUND: peer is not found
- ESP_ERR_ESPNOW_IF: current WiFi interface doesn’t match that of peer

`esp_err_t esp_now_add_peer (const esp_now_peer_info_t *peer)`
Add a peer to peer list.

参数 `peer` – peer information

返回
- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG: invalid argument
- ESP_ERR_ESPNOW_FULL: peer list is full
- ESP_ERR_ESPNOW_NO_MEM: out of memory
- ESP_ERR_ESPNOW_EXIST: peer has existed

`esp_err_t esp_now_del_peer (const uint8_t *peer_addr)`
Delete a peer from peer list.

参数 `peer_addr` – peer MAC address

返回
- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG: invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND: peer is not found

`esp_err_t esp_now_mod_peer (const esp_now_peer_info_t *peer)`
Modify a peer.

参数 `peer` – peer information

返回
- ESP_OK: succeed
- ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG: invalid argument
- ESP_ERR_ESPNOW_FULL: peer list is full

`esp_err_t esp_wifi_config_espnow_rate (wifi_interface_t ifx, wifi_phy_rate_t rate)`
Config ESPNOW rate of specified interface.

Attention 1. This API should be called after esp_wifi_start().

参数
- `ifx` – Interface to be configured.
- `rate` – Phy rate to be configured.

返回
- ESP_OK: succeed
Chapter 2. API Reference

- others: failed

```c
esp_err_t esp_now_get_peer (const uint8_t* peer_addr, esp_now_peer_info_t* peer)
```
Get a peer whose MAC address matches peer_addr from peer list.

- parameter
  - `peer_addr` - peer MAC address
  - `peer` - peer information

- return
  - ESP_OK: succeed
  - ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG: invalid argument
  - ESP_ERR_ESPNOW_NOT_FOUND: peer is not found

```c
esp_err_t esp_now_fetch_peer (bool from_head, esp_now_peer_info_t* peer)
```
Fetch a peer from peer list. Only return the peer which address is unicast, for the multicast/broadcast address, the function will ignore and try to find the next in the peer list.

- parameter
  - `from_head` - fetch from head of list or not
  - `peer` - peer information

- return
  - ESP_OK: succeed
  - ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG: invalid argument
  - ESP_ERR_ESPNOW_NOT_FOUND: peer is not found

```c
bool esp_now_is_peer_exist (const uint8_t* peer_addr)
```
Peer exists or not.

- parameter
  - `peer_addr` - peer MAC address

- return
  - true: peer exists
  - false: peer not exists

```c
esp_err_t esp_now_get_peer_num (esp_now_peer_num_t* num)
```
Get the number of peers.

- parameter
  - `num` - number of peers

- return
  - ESP_OK: succeed
  - ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG: invalid argument

```c
esp_err_t esp_now_set_pmk (const uint8_t* pmk)
```
Set the primary master key.

**Attention**  1. primary master key is used to encrypt local master key

- parameter
  - `pmk` - primary master key

- return
  - ESP_OK: succeed
  - ESP_ERR_ESPNOW_NOT_INIT: ESPNOW is not initialized
  - ESP_ERR_ESPNOW_ARG: invalid argument

```c
esp_err_t esp_now_set_wake_window (uint16_t window)
```
Set wake window for esp_now to wake up in interval unit.
Attention 1. This configuration could work at connected status. When ESP_WIFI_STA_DISCONNECTED_PM_ENABLE is enabled, this configuration could work at disconnected status.

Attention 2. Default value is the maximum.

参数  window  —  Milliseconds would the chip keep waked each interval, from 0 to 65535.

返回
  •  ESP_OK : succeed
  •  ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

**Structures**

`struct esp_now_peer_info`  
ESPNOW peer information parameters.

**Public Members**

`uint8_t peer_addr[ESP_NOW_ETH_ALEN]`  
ESPNOW peer MAC address that is also the MAC address of station or softap

`uint8_t lmk[ESP_NOW_KEY_LEN]`  
ESPNOW peer local master key that is used to encrypt data

`uint8_t channel`  
Wi-Fi channel that peer uses to send/receive ESPNOW data. If the value is 0, use the current channel which station or softap is on. Otherwise, it must be set as the channel that station or softap is on.

`wifi_interface_t ifidx`  
Wi-Fi interface that peer uses to send/receive ESPNOW data

`bool encrypt`  
ESPNOW data that this peer sends/receives is encrypted or not

`void *priv`  
ESPNOW peer private data

`struct esp_now_peer_num`  
Number of ESPNOW peers which exist currently.

**Public Members**

`int total_num`  
Total number of ESPNOW peers, maximum value is ESP_NOW_MAX_TOTAL_PEER_NUM

`int encrypt_num`  
Number of encrypted ESPNOW peers, maximum value is ESP_NOW_MAX_ENCRYPT_PEER_NUM
Chapter 2. API 参考

Macros

**ESP_ERR_ESPNOW_BASE**
ESPNOW error number base.

**ESP_ERR_ESPNOW_NOT_INIT**
ESPNOW is not initialized.

**ESP_ERR_ESPNOW_ARG**
Invalid argument

**ESP_ERR_ESPNOW_NO_MEM**
Out of memory

**ESP_ERR_ESPNOW_FULL**
ESPNOW peer list is full

**ESP_ERR_ESPNOW_NOT_FOUND**
ESPNOW peer is not found

**ESP_ERR_ESPNOW_INTERNAL**
Internal error

**ESP_ERR_ESPNOW_EXIST**
ESPNOW peer has existed

**ESP_ERR_ESPNOW_IF**
Interface error

**ESP_NOW_ETH_ALEN**
Length of ESPNOW peer MAC address

**ESP_NOW_KEY_LEN**
Length of ESPNOW peer local master key

**ESP_NOW_MAX_TOTAL_PEER_NUM**
Maximum number of ESPNOW total peers

**ESP_NOW_MAX_ENCRYPT_PEER_NUM**
Maximum number of ESPNOW encrypted peers

**ESP_NOW_MAX_DATA_LEN**
Maximum length of ESPNOW data which is sent very time

Type Definitions

typedef struct esp_now_peer_info esp_now_peer_info_t
ESPNOW peer information parameters.

typedef struct esp_now_peer_num esp_now_peer_num_t
Number of ESPNOW peers which exist currently.
typedef void (*esp_now_recv_cb_t)(const uint8_t*mac_addr, const uint8_t*data, int data_len)
Callback function of receiving ESPNOW data.

Param mac_addr peer MAC address
Param data received data
Param data_len length of received data

typedef void (*esp_now_send_cb_t)(const uint8_t*mac_addr, esp_now_send_status_t status)
Callback function of sending ESPNOW data.

Param mac_addr peer MAC address
Param status status of sending ESPNOW data (succeed or fail)

Enumerations

enum esp_now_send_status_t
Status of sending ESPNOW data.

Values:

enumerator ESP_NOW_SEND_SUCCESS
Send ESPNOW data successfully

enumerator ESP_NOW_SEND_FAIL
Send ESPNOW data fail

ESP-WIFI-MESH 編程指南

这是 ESP-WIFI-MESH 的编程指南，包括 API 参考和编码示例。本指南分为以下部分：

1. ESP-WIFI-MESH 编程模型
2. 编写 ESP-WIFI-MESH 应用程序
3. 自组网
4. 应用实例
5. API 参考

有关 ESP-WIFI-MESH 协议的文档，请见 ESP-WIFI-MESH API 指南。有关 ESP-WIFI-MESH 开发框架的更多内容，请见 ESP-WIFI-MESH 开发框架。

ESP-WIFI-MESH 编程模型

软件栈 ESP-WIFI-MESH 软件栈基于 Wi-Fi 驱动程序和 FreeRTOS 构建，某些情况下（如根节点）也会使用 LwIP 软件栈。下图展示了 ESP-WIFI-MESH 软件栈。

系统事件 应用程序可通过 ESP-WIFI-MESH 事件与 ESP-WIFI-MESH 交互。由于 ESP-WIFI-MESH 构建在 Wi-Fi 软件栈之上，因此也可以通过 Wi-Fi 事件任务与 Wi-Fi 驱动程序进行交互。下图展示了 ESP-WIFI-MESH 应用程序中各种系统事件的接口。

mesh_event_id_t 定义了所有可能的 ESP-WIFI-MESH 事件，并且可以指示父节点和子节点的连接或断开等事件。应用程序如需使用 ESP-WIFI-MESH 事件，则必须通过 esp_event_handler_register() 将 Mesh 事件处理程序注册在默认事件任务中。注册完成后，ESP-WIFI-MESH 事件将包含与应用程序所有相关事件相关的处理程序。
Chapter 2. API 参考

图 2: ESP-WIFI-MESH 软件栈

图 3: ESP-WIFI-MESH 系统事件交互

Mesh 事件的典型应用场景包括：使用 MESH_EVENT_PARENT_CONNECTED 和 MESH_EVENT_CHILD_CONNECTED 事件来指示节点何时可以分别开始传输上行和下行的数据。同样，也可以使用 IP_EVENT_STA_GOT_IP 和 IP_EVENT_STA_LOST_IP 事件来指示根节点何时可以向外 IP 网络传输数据。

警告：在自组网模式下使用 ESP-WIFI-MESH 时，用户必须确保不得调用 Wi-Fi API。原因在于：自组网模式将在内部调用 Wi-Fi API 实现连接/断开/扫描等操作。此时，如果外部应用程序调用 Wi-Fi API（包括来自回调函数和 Wi-Fi 事件处理程序的调用）都可能会干扰 ESP-WIFI-MESH 的自组网行为。因此，用户不应该在 esp_mesh_start() 和 esp_mesh_stop() 之间调用 Wi-Fi API。

LwIP & ESP-WIFI-MESH 应用程序无需通过 LwIP 层即可直接访问 ESP-WIFI-MESH 软件栈。LwIP 层仅在根节点和外部 IP 网络的数据发送与接收时会用到。但是，由于每个节点都有可能成为根节点（由于自组网节点选择机制的存在），每个节点仍必须初始化 LwIP 软件栈。

可成为根节点的每个节点都需要通过调用 esp_netif_init() 来初始化 LwIP 软件栈。为了防止非根节点访问 LwIP，应用程序不应使用 esp_netif API 创建或注册任何网络接口。

ESP-WIFI-MESH 的根节点必须与路由器连接。因此，当一个节点成为根节点时，该节点对应的应用程序必须启动 DHCP 客户端服务并立即获取 IP 地址。这样做将允许其他节点开始用于从外部 IP 网络发往/接收数据包。但是，如果使用静态 IP 设置，则不需要执行此步骤。

编写 ESP-WIFI-MESH 应用程序 ESP-WIFI-MESH 在正常启动前必须先初始化 LwIP 和 Wi-Fi 软件栈。下文代码展示了 ESP-WIFI-MESH 在开始自身初始化前必须完成的步骤。
```c
ESP_ERROR_CHECK(esp_netif_init());

/* 开始初始化 */
ESP_ERROR_CHECK(esp_event_loop_create_default());

/* Wi-Fi 初始化 */
wifi_init_config_t config = WIFI_INIT_CONFIG_DEFAULT();
ESP_ERROR_CHECK(esp_wifi_init(&config));

/* 注册 IP 事件处理器 */
ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &ip_event_handler, NULL));
ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_FLASH));
ESP_ERROR_CHECK(esp_wifi_start());

LwIP Wi-Fi 很可能 ESP-WIFI-MESH

ESP-WIFI-MESH

1. 初始化 Mesh
2. 配置 ESP-WIFI-MESH 网络
3. 启动 Mesh

初始化 Mesh 下方代码片段展示如何初始化 ESP-WIFI-MESH。

```c
/* Mesh 初始化 */
ESP_ERROR_CHECK(esp_mesh_init());

/* 注册 mesh 事件处理器 */
ESP_ERROR_CHECK(esp_event_handler_register(MESH_EVENT, ESP_EVENT_ANY_ID, &mesh_event_handler, NULL));
``` 

配置 ESP-WIFI-MESH 网络 ESP-WIFI-MESH 可通过 esp_mesh_set_config() 进行配置，并使用 mesh_cfg_t 结构体传递参数。该结构体包含以下 ESP-WIFI-MESH 的配置参数:

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel (信道)</td>
<td>1 到 14 信道</td>
</tr>
<tr>
<td>Mesh ID</td>
<td>ESP-WIFI-MESH 网络的 ID，见 mesh_addr_t。</td>
</tr>
<tr>
<td>Router (路由器)</td>
<td>路由器配置，见 mesh_router_t。</td>
</tr>
<tr>
<td>Mesh AP</td>
<td>Mesh AP 配置，见 mesh_ap_cfg_t。</td>
</tr>
<tr>
<td>Crypto Functions (加密函数)</td>
<td>Mesh IE 的加密函数，见 mesh_crypto_funcs_t。</td>
</tr>
</tbody>
</table>

下方代码片段展示如何配置 ESP-WIFI-MESH。

```c
/* 默认启用 Mesh IE 加密 */
mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();

/* Mesh ID */
memcpy((uint8_t *)&cfg.mesh_id, MESH_ID, 6);

/* 信道（需与路由器信道匹配）*/
cfg.channel = CONFIG_MESH_CHANNEL;

/* 路由器 */
cfg.router.ssid_len = strlen(CONFIG_MESH_ROUTER_SSID);
memcpy((uint8_t *)&cfg.router.ssid, CONFIG_MESH_ROUTER_SSID, cfg.router.ssid_len);
memcpy((uint8_t *)&cfg.router.password, CONFIG_MESH_ROUTER_PASSWD, strlen(CONFIG_MESH_ROUTER_PASSWD));

/* Mesh softAP */
cfg.mesh_ap.max_connection = CONFIG_MESH_AP_CONNECTIONS;
memcpy((uint8_t *)&cfg.mesh_ap.password, CONFIG_MESH_AP_PASSWD, strlen(CONFIG_MESH_AP_PASSWD));
ESP_ERROR_CHECK(esp_mesh_set_config(&cfg));
```
启动 Mesh 下方代码片段展示了如何启动 ESP-WIFI-MESH。

```c
/* 启动 Mesh */
ESP_ERROR_CHECK(esp_mesh_start());
```

启动 ESP-WIFI-MESH 后，应用程序应检查 ESP-WIFI-MESH 事件。以确定它何时连接到网络。连接后，应用程序可使用 `esp_mesh_send()` 和 `esp_mesh_recv()` 在 ESP-WIFI-MESH 网络中发送、接收数据包。

自组网 自组网是 ESP-WIFI-MESH 的功能之一，允许节点自动扫描/选择/连接/重新连接到其他节点和路由器。此功能允许 ESP-WIFI-MESH 网络具有很高的自主性，可适应变化的动态网络拓扑结构和环境。启用自组网功能后，ESP-WIFI-MESH 网络中的节点能够自主完成以下操作：

- 选择或选举根节点（见 [建立网络 中的 自动根节点选择]）
- 选择首席的父节点（见 [建立网络 中的 父节点选择]
- 网络断开时自动重新连接（见 [管理网络 中的 中间父节点失败]

启用自组网功能后，ESP-WIFI-MESH 软件栈将内部调用 Wi-Fi API。因此，在启用自组网功能时，应用层不得调用 Wi-Fi API，否则会干扰 ESP-WIFI-MESH 的工作。

开关自组网 应用程序可以在运行时通过调用 `esp_mesh_set_self_organized()` 函数，启用或禁用自组网功能。该函数具有以下两个参数：

- bool enable 指定启用或禁用自组网功能。
- bool select_parent 指定在启用自组网功能时是否应选择新的父节点。根据节点类型和节点当前状态，选择新的父节点具有不同的作用。在禁用自组网功能时，此参数不使用。

禁用自组网 下方代码片段展示了如何禁用自组网功能。

```c
//禁用自组网
esp_mesh_set_self_organized(false, false);
```

ESP-WIFI-MESH 将在禁用自组网时尝试维护节点的当前 Wi-Fi 状态。

- 如果节点先前已连接到其他节点，则将保持连接。
- 如果节点先前已断开连接并且正在扫描父节点或路由器，则将停止扫描。
- 如果节点先前尝试重新连接到父节点或路由器，则将停止重新连接。

启用自组网 ESP-WIFI-MESH 将尝试在启用自组网时保持节点的当前 Wi-Fi 状态。但是，根据节点类型以及是否选择了新的父节点，节点的 Wi-Fi 状态可能会发生变化。下表显示了启用自组网的效果。

<table>
<thead>
<tr>
<th>是否选择父节点</th>
<th>是否为根结点</th>
<th>作用</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>已连接到父节点的节点将保持连接。 之后扫描父节点的节点将停止扫描。调用 <code>esp_mesh_connect()</code> 重新启动。</td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>已连接到路由器的根节点将保持连接。 从路由器断开的根结点需调用 <code>esp_mesh_connect()</code> 进行重连。</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td>没有父节点的节点将自动选择首选父节点并连接。 从路由器断开的根结点需调用 <code>esp_mesh_connect()</code> 进行重连。</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>已连接到父节点的节点将断开连接，重新选择首选父节点并进行重连。 从路由器断开的根结点需调用 <code>esp_mesh_connect()</code> 进行重连。</td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>根节点在连接至父节点前必须放弃“根结点”的角色。因此，根节点将断开与路由器和所有子节点的连接，选择首选父节点并进行连接。</td>
</tr>
</tbody>
</table>

下方代码片段展示了如何启用自组网功能。

```c
//启用自组网，选择一个新的父节点
esp_mesh_set_self_organized(true, true);
```

(下页继续)
调用 Wi-Fi API 在有些情况下，应用程序可能希望在使用 ESP-WIFI-MESH 期间调用 Wi-Fi API。例如，应用程序可能需要手动扫描附近的接入点（AP）。但应用程序调用任何 Wi-Fi API 之前，必须先禁用自组网。否则，ESP-WIFI-MESH 软件栈可能会同时调用 Wi-Fi API，进而影响应用程序的正常使用。

应用程序不应在 esp_mesh_set_self_organized() 之间调用 Wi-Fi API。下方代码片段展示了应用程序如何在 ESP-WIFI-MESH 运行期间安全地调用 esp_wifi_scan_start()。

```c
//禁用自组网
esp_mesh_set_self_organized(0, 0);
//停止任何正在进行的扫描
esp_wifi_scan_stop();
//手动启动扫描运行完成时自动停止
esp_wifi_scan_start();
//进程扫描结果

... //如果仍为连接状态，则重新启用自组网
esp_mesh_set_self_organized(1, 0);

... //如果不为根节点且未连接，则重新启用自组网
esp_mesh_set_self_organized(1, 1);

... //如果为根节点且未连接，则重新启用
esp_mesh_set_self_organized(1, 0); //不选择新的父节点
esp_mesh_connect(); //手动重新连接到路由器
```

应用实例 ESP-IDF 包含以下 ESP-WIFI-MESH 示例项目：

内部通信示例 展示了如何搭建 ESP-WIFI-MESH 网络，并让根节点向网络中的每个节点发送数据包。

手动连网示例 展示了如何在禁止自组网功能的情况下使用 ESP-WIFI-MESH。此示例展示了如何对节点进行编程，以手动扫描潜在父节点的列表，并根据自定义标准选择父节点。

API 参考

Header File

- components/esp_wifi/include/esp_mesh.h

Functions

- esp_err_t esp_mesh_init (void)
  
  Mesh initialization.
• Check whether Wi-Fi is started.
• Initialize mesh global variables with default values.

**Attention** This API shall be called after Wi-Fi is started.

```
return
  • ESP_OK
  • ESP_FAIL
```

`esp_err_t esp_mesh_deinit (void)`
Mesh de-initialization.

- Release resources and stop the mesh

```
return
  • ESP_OK
  • ESP_FAIL
```

`esp_err_t esp_mesh_start (void)`
Start mesh.

• Initialize mesh IE.
• Start mesh network management service.
• Create TX and RX queues according to the configuration.
• Register mesh packets receive callback.

**Attention** This API shall be called after mesh initialization and configuration.

```
return
  • ESP_OK
  • ESP_FAIL
  • ESP_ERR_MESH_NOT_INIT
  • ESP_ERR_MESH_NOT_CONFIG
  • ESP_ERR_MESH_NO_MEMORY
```

`esp_err_t esp_mesh_stop (void)`
Stop mesh.

• Deinitialize mesh IE.
• Disconnect with current parent.
• Disassociate all currently associated children.
• Stop mesh network management service.
• Unregister mesh packets receive callback.
• Delete TX and RX queues.
• Release resources.
• Restore Wi-Fi softAP to default settings if Wi-Fi dual mode is enabled.
• Set Wi-Fi Power Save type to WIFI_PS_NONE.

```
return
  • ESP_OK
  • ESP_FAIL
```
**esp_err_t esp_mesh_send**  
(const *to, const *data, int flag, const *opt[], int opt_count)

Send a packet over the mesh network.

- Send a packet to any device in the mesh network.
- Send a packet to external IP network.

**Attention** This API is not reentrant.

**参数**
- **to** [in] the address of the final destination of the packet
  - If the packet is to the root, set this parameter to NULL.
  - If the packet is to an external IP network, set this parameter to the IPv4:PORT combination. This packet will be delivered to the root firstly, then the root will forward this packet to the final IP server address.
- **data** [in] pointer to a sending mesh packet
  - Field size should not exceed MESH_MPS. Note that the size of one mesh packet should not exceed MESH_MTU.
  - Field proto should be set to data protocol in use (default is MESH_PROTO_BIN for binary).
  - Field tos should be set to transmission tos (type of service) in use (default is MESH_TOS_P2P for point-to-point reliable).
- **flag** [in] bitmap for data sent
  - Speed up the route search
    - If the packet is to the root and “to” parameter is NULL, set this parameter to 0.
    - If the packet is to an internal device, MESH_DATA_P2P should be set.
    - If the packet is to the root (“to” parameter isn’t NULL) or to external IP network, MESH_DATA_TODS should be set.
    - If the packet is from the root to an internal device, MESH_DATA_FROMDS should be set.
    - Specify whether this API is block or non-block, block by default
      - If needs non-blocking, MESH_DATA_NONBLOCK should be set. Otherwise, may use esp_mesh_send_block_time() to specify a blocking time.
    - In the situation of the root change, MESH_DATA_DROP identifies this packet can be dropped by the new root for upstream data to external IP network, we try our best to avoid data loss caused by the root change, but there is a risk that the new root is running out of memory because most of memory is occupied by the pending data which isn’t read out in time by esp_mesh_recv_toDS(). Generally, we suggest esp_mesh_recv_toDS() is called after a connection with IP network is created. Thus data outgoing to external IP network via socket is just from reading esp_mesh_recv_toDS() which avoids unnecessary memory copy.
- **opt** [in] options
  - In case of sending a packet to a certain group, MESH_OPT_SEND_GROUP is a good choice. In this option, the value field should be set to the target receiver addresses in this group.
  - Root sends a packet to an internal device, this packet is from external IP network in case the receiver device responds this packet, MESH_OPT_RECV_DS_ADDR is required to attach the target DS address.
- **opt_count** [in] option count
  - Currently, this API only takes one option, so opt_count is only supported to be 1.

**返回**
- ESP_OK
- ESP_FAIL
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_DISCONNECTED
• ESP_ERR_MESH_OPT_UNKNOWN
• ESP_ERR_MESH_EXCEED_MTU
• ESP_ERR_MESH_NO_MEMORY
• ESP_ERR_MESH_TIMEOUT
• ESP_ERR_MESH_QUEUE_FULL
• ESP_ERR_MESH_NO_ROUTE_FOUND
• ESP_ERR_MESH_DISCARD

`esp_err_t esp_mesh_send_block_time (uint32_t time_ms)`
Set blocking time of esp_mesh_send()

**Attention** This API shall be called before mesh is started.

参数  `time_ms` -in blocking time of esp_mesh_send(), unit: ms
返回  ESP_OK

`esp_err_t esp_mesh_recv (mesh_addr_t *from, mesh_data_t *data, int timeout_ms, int *flag, mesh_opt_t *opt[], int opt_count)`
Receive a packet targeted to self over the mesh network.

flag could be MESH_DATA_FROMDS or MESH_DATA_TODS.

**Attention** Mesh RX queue should be checked regularly to avoid running out of memory.
• Use esp_mesh_get_rx_pending() to check the number of packets available in the queue waiting to be received by applications.

参数
• `from` -out the address of the original source of the packet
• `data` -out pointer to the received mesh packet
  -- Field proto is the data protocol in use. Should follow it to parse the received data.
  -- Field tos is the transmission tos (type of service) in use.
• `timeout_ms` -in wait time if a packet isn’t immediately available (0:no wait, port-MAX_DELAY:wait forever)
• `flag` -out bitmap for data received
  -- MESH_DATA_FROMDS represents data from external IP network
  -- MESH_DATA_TODS represents data directed upward within the mesh network
• `opt` -out options desired to receive
  -- MESH_OPT_RECV_DS_ADDR attaches the DS address
• `opt_count` -in option count desired to receive
  -- Currently, this API only takes one option, so opt_count is only supported to be 1.

返回
• ESP_OK
• ESP_ERR_MESH_ARGUMENT
• ESP_ERR_MESH_NOT_START
• ESP_ERR_MESH_TIMEOUT
• ESP_ERR_MESH_DISCARD

`esp_err_t esp_mesh_recv_toDS (mesh_addr_t *from, mesh_addr_t *to, mesh_data_t *data, int timeout_ms, int *flag, mesh_opt_t *opt[], int opt_count)`
Receive a packet targeted to external IP network.

• Root uses this API to receive packets destined to external IP network
• Root forwards the received packets to the final destination via socket.
• If no socket connection is ready to send out the received packets and this esp_mesh_recv_toDS() hasn’t been called by applications, packets from the whole mesh network will be pending in toDS queue.

Use esp_mesh_get_rx_pending() to check the number of packets available in the queue waiting to be received by applications in case of running out of memory in the root.

Using esp_mesh_set_xon_qsize() users may configure the RX queue size, default:32. If this size is too large, and esp_mesh_recv_toDS() isn’t called in time, there is a risk that a great deal of memory is occupied by the pending packets. If this size is too small, it will impact the efficiency on upstream. How to decide this value depends on the specific application scenarios.

flag could be MESH_DATA_TODS.

Attention This API is only called by the root.

参数

• from - [out] the address of the original source of the packet
• to - [out] the address contains remote IP address and port (IPv4:PORT)
• data - [out] pointer to the received packet
  – Contain the protocol and applications should follow it to parse the data.
• timeout_ms - [in] wait time if a packet isn’t immediately available (0:no wait, port-MAX_DELAY:wait forever)
• flag - [out] bitmap for data received
  – MESH_DATA_TODS represents the received data target to external IP network. Root shall forward this data to external IP network via the association with router.
• opt - [out] options desired to receive
• opt_count - [in] option count desired to receive

返回

• ESP_OK
• ESP_ERR_MESH_ARGUMENT
• ESP_ERR_MESH_NOT_START
• ESP_ERR_MESH_TIMEOUT
• ESP_ERR_MESH_DISCONNECT
• ESP_ERR_MESH_RECV_RELEASE

esp_err_t esp_mesh_set_config(const mesh_cfg_t *config)

Set mesh stack configuration.

• Use MESH_INIT_CONFIG_DEFAULT() to initialize the default values, mesh IE is encrypted by default.
• Mesh network is established on a fixed channel (1-14).
• Mesh event callback is mandatory.
• Mesh ID is an identifier of an MBSS. Nodes with the same mesh ID can communicate with each other.
• Regarding to the router configuration, if the router is hidden, BSSID field is mandatory.

If BSSID field isn’t set and there exists more than one router with same SSID, there is a risk that more roots than one connected with different BSSID will appear. It means more than one mesh network is established with the same mesh ID.

Root conflict function could eliminate redundant roots connected with the same BSSID, but couldn’t handle roots connected with different BSSID. Because users might have such requirements of setting up routers with same SSID for the future replacement. But in that case, if the above situations happen, please make sure applications implement forward functions on the root to guarantee devices in different mesh networks can communicate with each other. max_connection of mesh softAP is limited by the max number of Wi-Fi softAP supported (max:10).

Attention This API shall be called before mesh is started after mesh is initialized.
### Chapter 2. API 参考

#### esp_mesh_get_config

**Function:**
Get mesh stack configuration.

**Syntax:**
```c
esp_err_t esp_mesh_get_config(mesh_cfg_t *config)
```

**Parameters:**
- `config` [in] pointer to mesh stack configuration

**Return Codes:**
- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

#### esp_mesh_set_router

**Function:**
Get router configuration.

**Syntax:**
```c
esp_err_t esp_mesh_set_router(const mesh_router_t *router)
```

**Parameters:**
- `router` [in] pointer to router configuration

**Return Codes:**
- ESP_OK
- ESP_ERR_MESH_ARGUMENT

**Attention:** This API is used to dynamically modify the router configuration after mesh is configured.

#### esp_mesh_get_router

**Function:**
Get router configuration.

**Syntax:**
```c
esp_err_t esp_mesh_get_router(mesh_router_t *router)
```

**Parameters:**
- `router` [out] pointer to router configuration

**Return Codes:**
- ESP_OK
- ESP_ERR_MESH_ARGUMENT

#### esp_mesh_set_id

**Function:**
Set mesh network ID.

**Syntax:**
```c
esp_err_t esp_mesh_set_id(const mesh_addr_t *id)
```

**Parameters:**
- `id` [in] pointer to mesh network ID

**Return Codes:**
- ESP_OK
- ESP_ERR_MESH_ARGUMENT: invalid argument

**Attention:** This API is used to dynamically modify the mesh network ID.

#### esp_mesh_get_id

**Function:**
Get mesh network ID.

**Syntax:**
```c
esp_err_t esp_mesh_get_id(mesh_addr_t *id)
```

**Parameters:**
- `id` [out] pointer to mesh network ID

**Return Codes:**
- ESP_OK
- ESP_ERR_MESH_ARGUMENT

#### esp_mesh_set_type

**Function:**
Designate device type over the mesh network.

**Syntax:**
```c
esp_err_t esp_mesh_set_type(mesh_type_t type)
```

**Parameters:**
- `type` device type

- MESH_IDLE: designates a device as a self-organized node for a mesh network
- MESH_ROOT: designates the root node for a mesh network
- MESH_LEAF: designates a device as a standalone Wi-Fi station that connects to a parent
- MESH_STA: designates a device as a standalone Wi-Fi station that connects to a router
mesh_type_t esp_mesh_get_type (void)
Get device type over mesh network.

Attention This API shall be called after having received the event MESH_EVENT_PARENT_CONNECTED.

返回 mesh type

esp_err_t esp_mesh_set_max_layer (int max_layer)
Set network max layer value.

- for tree topology, the max is 25.
- for chain topology, the max is 1000.
- Network max layer limits the max hop count.

Attention This API shall be called before mesh is started.

参数 max_layer -[in] max layer value
返回
- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

int esp_mesh_get_max_layer (void)
Get max layer value.

返回 max layer value

esp_err_t esp_mesh_set_ap_password (const uint8_t *pwd, int len)
Set mesh softAP password.

Attention This API shall be called before mesh is started.

参数 pwd -[in] pointer to the password
len -[in] password length
返回
- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

esp_err_t esp_mesh_set_ap_authmode (wifi_auth_mode_t authmode)
Set mesh softAP authentication mode.

Attention This API shall be called before mesh is started.

参数 authmode -[in] authentication mode
返回
**Chapter 2. API 参考**

- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

`wifi_auth_mode_t esp_mesh_get_ap_authmode (void)`
Get mesh softAP authentication mode.

- 类型 authentication mode

`esp_err_t esp_mesh_set_ap_connections (int connections)`
Set mesh max connection value.

- Set mesh softAP max connection = mesh max connection + non-mesh max connection

**Attention** This API shall be called before mesh is started.

**参数**
- connections - [in] the number of max connections

- 返回
  - ESP_OK
  - ESP_ERR_MESH_ARGUMENT

`int esp_mesh_get_ap_connections (void)`
Get mesh max connection configuration.

- 返回 the number of mesh max connections

`int esp_mesh_get_non_mesh_connections (void)`
Get non-mesh max connection configuration.

- 返回 the number of non-mesh max connections

`int esp_mesh_get_layer (void)`
Get current layer value over the mesh network.

**Attention** This API shall be called after having received the event MESH_EVENT_PARENT_CONNECTED.

- 返回 layer value

`esp_err_t esp_mesh_get_parent_bssid (mesh_addr_t *bssid)`
Get the parent BSSID.

**Attention** This API shall be called after having received the event MESH_EVENT_PARENT_CONNECTED.

**参数**
- bssid - [out] pointer to parent BSSID

- 返回
  - ESP_OK
  - ESP_FAIL

`bool esp_mesh_is_root (void)`
Return whether the device is the root node of the network.

- 返回 true/false
**esp_err_t esp_mesh_set_self_organized** (bool enable, bool select_parent)

Enable/disable self-organized networking.

- Self-organized networking has three main functions: select the root node; find a preferred parent; initiate reconnection if a disconnection is detected.
- Self-organized networking is enabled by default.
- If self-organized is disabled, users should set a parent for the device via esp_mesh_set_parent().

**Attention** This API is used to dynamically modify whether to enable the self organizing.

**参数**
- **enable**  [in] enable or disable self-organized networking
- **select_parent**  [in] Only valid when self-organized networking is enabled.
  - if select_parent is set to true, the root will give up its mesh root status and search for a new parent like other non-root devices.

**返回**
- ESP_OK
- ESP_FAIL

**bool esp_mesh_get_self_organized** (void)

Return whether enable self-organized networking or not.

**返回** true/false

**esp_err_t esp_mesh_waive_root** (const mesh_vote_t *vote, int reason)

Cause the root device to give up (waive) its mesh root status.

- A device is elected root primarily based on RSSI from the external router.
- If external router conditions change, users can call this API to perform a root switch.
- In this API, users could specify a desired root address to replace itself or specify an attempts value to ask current root to initiate a new round of voting. During the voting, a better root candidate would be expected to find to replace the current one.
- If no desired root candidate, the vote will try a specified number of attempts (at least 15). If no better root candidate is found, keep the current one. If a better candidate is found, the new better one will send a root switch request to the current root, current root will respond with a root switch acknowledgment.
- After that, the new candidate will connect to the router to be a new root, the previous root will disconnect with the router and choose another parent instead.

Root switch is completed with minimal disruption to the whole mesh network.

**Attention** This API is only called by the root.

**参数**
- **vote**  [in] vote configuration
  - If this parameter is set NULL, the vote will perform the default 15 times.
  - Field percentage threshold is 0.9 by default.
  - Field is_rc_specified shall be false.
  - Field attempts shall be at least 15 times.
- **reason**  [in] only accept MESH_VOTE_REASON_ROOT_INITIATED for now

**返回**
- ESP_OK
- ESP_ERR_MESH_QUEUE_FULL
- ESP_ERR_MESH_DISCARD
- ESP_FAIL
Chapter 2. API 参考

**esp_err_t esp_mesh_set_vote_percentage (float percentage)**
Set vote percentage threshold for approval of being a root (default: 0.9)

- During the networking, only obtaining vote percentage reaches this threshold, the device could be a root.

**Attention** This API shall be called before mesh is started.

<table>
<thead>
<tr>
<th>参数</th>
<th>percentage</th>
<th>[in] vote percentage threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>返回</td>
<td>ESP_OK</td>
<td>ESP_FAIL</td>
</tr>
</tbody>
</table>

**float esp_mesh_get_vote_percentage (void)**
Get vote percentage threshold for approval of being a root.

| 返回 | percentage threshold |

**esp_err_t esp_mesh_set_ap_assoc_expire (int seconds)**
Set mesh softAP associate expired time (default: 10 seconds)

- If mesh softAP hasn’t received any data from an associated child within this time, mesh softAP will take this child inactive and disassociate it.
- If mesh softAP is encrypted, this value should be set a greater value, such as 30 seconds.

<table>
<thead>
<tr>
<th>参数</th>
<th>seconds</th>
<th>[in] the expired time</th>
</tr>
</thead>
<tbody>
<tr>
<td>返回</td>
<td>ESP_OK</td>
<td>ESP_FAIL</td>
</tr>
</tbody>
</table>

**int esp_mesh_get_ap_assoc_expire (void)**
Get mesh softAP associate expired time.

| 返回 | seconds |

**int esp_mesh_get_total_node_num (void)**
Get total number of devices in current network (including the root).

**Attention** The returned value might be incorrect when the network is changing.

| 返回 | total number of devices (including the root) |

**int esp_mesh_get_routing_table_size (void)**
Get the number of devices in this device’s sub-network (including self)

| 返回 | the number of devices over this device’s sub-network (including self) |

**esp_err_t esp_mesh_get_routing_table (mesh_addr_t *mac, int len, int *size)**
Get routing table of this device’s sub-network (including itself)

<table>
<thead>
<tr>
<th>参数</th>
<th>mac [out] pointer to routing table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>len [in] routing table size (in bytes)</td>
</tr>
<tr>
<td></td>
<td>size [out] pointer to the number of devices in routing table (including itself)</td>
</tr>
<tr>
<td>返回</td>
<td>ESP_OK</td>
</tr>
<tr>
<td></td>
<td>ESP_ERR_MESH_ARGUMENT</td>
</tr>
</tbody>
</table>
**esp_err_t esp_mesh_post_toDS_state (bool reachable)**

Post the toDS state to the mesh stack.

**Attention** This API is only for the root.

- **reachable** – [in] this state represents whether the root is able to access external IP network
- **ESP_OK**
- **ESP_FAIL**

**esp_err_t esp_mesh_get_tx_pending (mesh_tx_pending_t *pending)**

Return the number of packets pending in the queue waiting to be sent by the mesh stack.

- **pending** – [out] pointer to the TX pending
- **ESP_OK**
- **ESP_FAIL**

**esp_err_t esp_mesh_get_rx_pending (mesh_rx_pending_t *pending)**

Return the number of packets available in the queue waiting to be received by applications.

- **pending** – [out] pointer to the RX pending
- **ESP_OK**
- **ESP_FAIL**

**int esp_mesh_available_txupQ_num (const mesh_addr_t *addr, uint32_t *xseqno_in)**

Return the number of packets could be accepted from the specified address.

- **addr** – [in] self address or an associate children address
- **xseqno_in** – [out] sequence number of the last received packet from the specified address
- **ESP_OK**
- **ESP_FAIL**

**esp_err_t esp_mesh_set_xon_qsize (int qsize)**

Set the number of queue.

**Attention** This API shall be called before mesh is started.

- **qsize** – [in] default:32 (min:16)
- **ESP_OK**
- **ESP_FAIL**

**int esp_mesh_get_xon_qsize (void)**

Get queue size.

- **ESP_OK**
- **ESP_FAIL**

**esp_err_t esp_mesh_allow_root_conflicts (bool allowed)**

Set whether allow more than one root existing in one network.

- **allowed** – [in] allow or not
- **ESP_OK**
- **ESP_WIFI_ERR_NOT_INIT**
- **ESP_WIFI_ERR_NOT_START**
bool esp_mesh_is_root_conflicts_allowed(void)
Check whether allow more than one root to exist in one network.

返回  true/false

esp_err_t esp_mesh_set_group_id (const mesh_addr_t *addr, int num)
Set group ID addresses.

参数
  • addr - [in] pointer to new group ID addresses
  • num - [in] the number of group ID addresses

返回
  • ESP_OK
  • ESP_MESH_ERR_ARGUMENT

esp_err_t esp_mesh_delete_group_id (const mesh_addr_t *addr, int num)
Delete group ID addresses.

参数
  • addr - [in] pointer to deleted group ID address
  • num - [in] the number of group ID addresses

返回
  • ESP_OK
  • ESP_MESH_ERR_ARGUMENT

int esp_mesh_get_group_num (void)
Get the number of group ID addresses.

返回  the number of group ID addresses

esp_err_t esp_mesh_get_group_list (mesh_addr_t *addr, int num)
Get group ID addresses.

参数
  • addr - [out] pointer to group ID addresses
  • num - [in] the number of group ID addresses

返回
  • ESP_OK
  • ESP_MESH_ERR_ARGUMENT

bool esp_mesh_is_my_group (const mesh_addr_t *addr)
Check whether the specified group address is my group.

返回  true/false

esp_err_t esp_mesh_set_capacity_num (int num)
Set mesh network capacity (max:1000, default:300)

Attention This API shall be called before mesh is started.

参数 num - [in] mesh network capacity

返回
  • ESP_OK
  • ESP_ERR_MESH_NOT_ALLOWED
  • ESP_MESH_ERR_ARGUMENT

int esp_mesh_get_capacity_num (void)
Get mesh network capacity.

返回  mesh network capacity
**esp_err_t esp_mesh_set_ie_crypto_funcs** (const mesh_crypto_funcs_t *crypto_funcs)

Set mesh IE crypto functions.

**Attention** This API can be called at any time after mesh is initialized.

- **参数** crypto_funcs - [in] crypto functions for mesh IE
  - If crypto_funcs is set to NULL, mesh IE is no longer encrypted.
  - **返回** ESP_OK

**esp_err_t esp_mesh_set_ie_crypto_key** (const char* key, int len)

Set mesh IE crypto key.

**Attention** This API can be called at any time after mesh is initialized.

- **参数**
  - **key** - [in] ASCII crypto key
  - **len** - [in] length in bytes, range: 8~64
- **返回**
  - ESP_OK
  - ESP_MESH_ERR_ARGUMENT

**esp_err_t esp_mesh_set_root_healing_delay** (int delay_ms)

Set delay time before starting root healing.

- **参数** delay_ms - [in] delay time in milliseconds
  - **返回** ESP_OK

**int esp_mesh_get_root_healing_delay** (void)

Get delay time before network starts root healing.

- **返回** delay time in milliseconds

**esp_err_t esp_mesh_set_root** (bool enable)

Enable network Fixed Root Setting.

- Enabling fixed root disable automatic election of the root node via voting.
- All devices in the network shall use the same Fixed Root Setting (enabled or disabled).
- If Fixed Root is enabled, users should make sure a root node is designated for the network.

- **参数** enable - [in] enable or not
  - **返回** ESP_OK
bool esp_mesh_is_root_fixed(void)

Check whether network Fixed Root Setting is enabled.

- Enable/disable network Fixed Root Setting by API esp_mesh_fix_root().
- Network Fixed Root Setting also changes with the “flag” value in parent networking IE.

返回 true/false

esp_err_t esp_mesh_set_parent(const wifi_config_t *parent, const mesh_addr_t *parent_mesh_id, mesh_type_t my_type, int my_layer)

Set a specified parent for the device.

Attention This API can be called at any time after mesh is configured.

参数
- parent - [in] parent configuration, the SSID and the channel of the parent are mandatory.
  - If the BSSID is set, make sure that the SSID and BSSID represent the same parent, otherwise the device will never find this specified parent.
- parent_mesh_id - [in] parent mesh ID,
  - If this value is not set, the original mesh ID is used.
- my_type - [in] mesh type
  - MESH_STA is not supported.
  - If the parent set for the device is the same as the router in the network configuration, then my_type shall set MESH_ROOT and my_layer shall set MESH_ROOT_LAYER.
- my_layer - [in] mesh layer
  - my_layer of the device may change after joining the network.
  - If my_type is set MESH_NODE, my_layer shall be greater than MESH_ROOT_LAYER.
  - If my_type is set MESH_LEAF, the device becomes a standalone Wi-Fi station and no longer has the ability to extend the network.

返回
- ESP_OK
- ESP_ERR_ARGUMENT
- ESP_ERR_MESH_NOT_CONFIG

esp_err_t esp_mesh_scan_get_ap_ie_len(int *len)

Get mesh networking IE length of one AP.

参数 len - [out] mesh networking IE length

返回
- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_WIFI_ARG
- ESP_ERR_WIFI_FAIL

esp_err_t esp_mesh_scan_get_ap_record(wifi_ap_record_t *ap_record, void *buffer)

Get AP record.

Attention Different from esp_wifi_scan_get_ap_records(), this API only gets one of APs scanned each time. See “manual_networking” example.

参数
- ap_record - [out] pointer to one AP record
- buffer - [out] pointer to the mesh networking IE of this AP

返回
Chapter 2. API 参数

- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_WIFI_ARG
- ESP_ERR_WIFI_FAIL

```c
esp_err_t esp_mesh_flush_upstream_packets (void)
```
Flush upstream packets pending in to_parent queue and to_parent_p2p queue.

返回
- ESP_OK

```c
esp_err_t esp_mesh_get_subnet_nodes_num (const mesh_addr_t *child_mac, int *nodes_num)
```
Get the number of nodes in the subnet of a specific child.

参数
- `child_mac` [in] an associated child address of this device
- `nodes_num` [out] pointer to the number of nodes in the subnet of a specific child

返回
- ESP_OK
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_ARGUMENT

```c
esp_err_t esp_mesh_get_subnet_nodes_list (const mesh_addr_t *child_mac, mesh_addr_t *nodes, int nodes_num)
```
Get nodes in the subnet of a specific child.

参数
- `child_mac` [in] an associated child address of this device
- `nodes` [out] pointer to nodes in the subnet of a specific child
- `nodes_num` [in] the number of nodes in the subnet of a specific child

返回
- ESP_OK
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_ARGUMENT

```c
esp_err_t esp_mesh_disconnect (void)
```
Disconnect from current parent.

返回
- ESP_OK

```c
esp_err_t esp_mesh_connect (void)
```
Connect to current parent.

返回
- ESP_OK

```c
esp_err_t esp_mesh_flush_scan_result (void)
```
Flush scan result.

返回
- ESP_OK

```c
esp_err_t esp_mesh_switch_channel (const uint8_t *new_bssid, int csa_newchan, int csa_count)
```
Cause the root device to add Channel Switch Announcement Element (CSA IE) to beacon.

- Set the new channel
- Set how many beacons with CSA IE will be sent before changing a new channel
- Enable the channel switch function
**Attention** This API is only called by the root.

**参数**
- **new_bssid [in]** the new router BSSID if the router changes
- **csa_newchan [in]** the new channel number to which the whole network is moving
- **csa_count [in]** channel switch period(beacon count), unit is based on beacon interval of its softAP, the default value is 15.

**返回**
- ESP_OK

```c
esp_err_t esp_mesh_get_router_bssid(uint8_t* router_bssid)
```
Get the router BSSID.

**参数**
- **router_bssid [out]** pointer to the router BSSID

**返回**
- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_WIFI_ARG

```c
int64_t esp_mesh_get_tsf_time(void)
```
Get the TSF time.

**返回**
- the TSF time

```c
esp_err_t esp_mesh_set_topology(esp_mesh_topology_t topo)
```
Set mesh topology. The default value is MESH_TOPO_TREE.

- MESH_TOPO_CHAIN supports up to 1000 layers

**Attention** This API shall be called before mesh is started.

**参数**
- **topo [in]** MESH_TOPO_TREE or MESH_TOPO_CHAIN

**返回**
- ESP_OK
- ESP_MESH_ERR_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

```c
esp_mesh_topology_t esp_mesh_get_topology(void)
```
Get mesh topology.

**返回**
- MESH_TOPO_TREE or MESH_TOPO_CHAIN

```c
esp_err_t esp_mesh_enable_ps(void)
```
Enable mesh Power Save function.

**Attention** This API shall be called before mesh is started.

**返回**
- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_MESH_NOT_ALLOWED

```c
esp_err_t esp_mesh_disable_ps(void)
```
Disable mesh Power Save function.

**Attention** This API shall be called before mesh is started.
Chapter 2. API 参考

返回
- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_MESH_NOT_ALLOWED

bool esp_mesh_is_ps_enabled (void)
Check whether the mesh Power Save function is enabled.

返回 true/false

bool esp_mesh_is_device_active (void)
Check whether the device is in active state.

- If the device is not in active state, it will neither transmit nor receive frames.

返回 true/false

esp_err_t esp_mesh_set_active_duty_cycle (int dev_duty, int dev_duty_type)
Set the device duty cycle and type.

- The range of dev_duty values is 1 to 100. The default value is 10.
- dev_duty = 100, the PS will be stopped.
- dev_duty is better to not less than 5.
- dev_duty_type could be MESH_PS_DEVICE_DUTY_REQUEST or MESH_PS_DEVICE_DUTY_DEMAND.
- If dev_duty_type is set to MESH_PS_DEVICE_DUTY_REQUEST, the device will use a nwk_duty provided by the network.
- If dev_duty_type is set to MESH_PS_DEVICE_DUTY_DEMAND, the device will use the specified dev_duty.

Attention This API can be called at any time after mesh is started.

参数
- dev_duty  [in] device duty cycle
- dev_duty_type  [in] device PS duty cycle type, not accept MESH_PS_NETWORK_DUTY_MASTER

返回
- ESP_OK
- ESP_FAIL

esp_err_t esp_mesh_get_active_duty_cycle (int *dev_duty, int *dev_duty_type)
Get device duty cycle and type.

参数
- dev_duty  [out] device duty cycle
- dev_duty_type  [out] device PS duty cycle type

返回
- ESP_OK

esp_err_t esp_mesh_set_network_duty_cycle (int nwk_duty, int duration_mins, int applied_rule)
Set the network duty cycle, duration and rule.

- The range of nwk_duty values is 1 to 100. The default value is 10.
- nwk_duty is the network duty cycle the entire network or the up-link path will use. A device that successfully sets the nwk_duty is known as a NWK-DUTY-MASTER.
- duration_mins specifies how long the specified nwk_duty will be used. Once duration_mins expires, the root will take over as the NWK-DUTY-MASTER. If an existing NWK-DUTY-MASTER leaves the network, the root will take over as the NWK-DUTY-MASTER again.
- duration_mins = (-1) represents nwk_duty will be used until a new NWK-DUTY-MASTER with a different nwk_duty appears.
- Only the root can set duration_mins to (-1).
- If applied_rule is set to MESH_PS_NETWORK_DUTY_APPLIED_ENTIRE, the nwk_duty will be used by the entire network.
- If applied_rule is set to MESH_PS_NETWORK_DUTY_APPLIED_UPLINK, the nwk_duty will only be used by the up-link path nodes.
- The root does not accept MESH_PS_NETWORK_DUTY_APPLIED_UPLINK.
- A nwk_duty with duration_mins(-1) set by the root is the default network duty cycle used by the entire network.

**Attention** This API can be called at any time after mesh is started.
- In self-organized network, if this API is called before mesh is started in all devices, (1)nwk_duty shall be set to the same value for all devices; (2)duration_mins shall be set to (-1); (3)applied_rule shall be set to MESH_PS_NETWORK_DUTY_APPLIED_ENTIRE; after the voted root appears, the root will become the NWK-DUTY-MASTER and broadcast the nwk_duty and its identity of NWK-DUTY-MASTER.
- If the root is specified (FIXED-ROOT), call this API in the root to provide a default nwk_duty for the entire network.
- After joins the network, any device can call this API to change the nwk_duty, duration_mins or applied_rule.

**参数**
- nwk_duty - [in] network duty cycle
- duration_mins -[in] duration (unit: minutes)
- applied_rule -[in] only support MESH_PS_NETWORK_DUTY_APPLIED_ENTIRE

**返回**
- ESP_OK
- ESP_FAIL

```c
esp_err_t esp_mesh_get_network_duty_cycle (int*nwk_duty, int*duration_mins, int *dev_duty_type, int *applied_rule)
```

Get the network duty cycle, duration, type and rule.

**参数**
- nwk_duty - [out] current network duty cycle
- duration_mins -[out] the duration of current nwk_duty
- dev_duty_type -[out] if it includes MESH_PS_DEVICE_DUTY_MASTER, this device is the current NWK-DUTY-MASTER.
- applied_rule -[out] MESH_PS_NETWORK_DUTY_APPLIED_ENTIRE

**返回**
- ESP_OK

```c
int esp_mesh_get_running_active_duty_cycle (void)
```

Get the running active duty cycle.

- The running active duty cycle of the root is 100.
- If duty type is set to MESH_PS_DEVICE_DUTY_REQUEST, the running active duty cycle is nwk_duty provided by the network.
- If duty type is set to MESH_PS_DEVICE_DUTY_DEMAND, the running active duty cycle is dev_duty specified by the users.
- In a mesh network, devices are typically working with a certain duty-cycle (transmitting, receiving and sleep) to reduce the power consumption. The running active duty cycle decides the amount of awake time within a beacon interval. At each start of beacon interval, all devices wake up, broadcast beacons,
and transmit packets if they do have pending packets for their parents or for their children. Note that Low-duty-cycle means devices may not be active in most of the time, the latency of data transmission might be greater.

返回 the running active duty cycle

```c
esp_err_t esp_mesh_ps_duty_signaling(int fwd_times)
```

Duty signaling.

参数 `fwd_times` - [in] the times of forwarding duty signaling packets

返回
- ESP_OK

### Unions

```c
union mesh_addr_t
   #include <esp_mesh.h> Mesh address.
```

**Public Members**

```c
uint8_t addr[6]
   mac address
```

```c
mip_t mip
   mip address
```

```c
union mesh_event_info_t
   #include <esp_mesh.h> Mesh event information.
```

**Public Members**

```c
mesh_event_channel_switch_t channel_switch
   channel switch
```

```c
mesh_event_child_connected_t child_connected
   child connected
```

```c
mesh_event_child_disconnected_t child_disconnected
   child disconnected
```

```c
mesh_event_routing_table_change_t routing_table
   routing table change
```

```c
mesh_event_connected_t connected
   parent connected
```

```c
mesh_event_disconnected_t disconnected
   parent disconnected
```
```
Chapter 2. API 参考

mesh_event_no_parent_found_t no_parent
  no parent found

mesh_event_layer_change_t layer_change
  layer change

mesh_event_toDS_state_t toDS_state
  toDS state, devices shall check this state firstly before trying to send packets to external IP network. This state indicates right now whether the root is capable of sending packets out. If not, devices had better to wait until this state changes to be MESH_TODS_REACHABLE.

mesh_event_vote_started_t vote_started
  vote started

mesh_event_root_address_t root_addr
  root address

mesh_event_root_switch_req_t switch_req
  root switch request

mesh_event_root_conflict_t root_conflict
  other powerful root

mesh_event_root_fixed_t root_fixed
  fixed root

mesh_event_scan_done_t scan_done
  scan done

mesh_event_network_state_t network_state
  network state, such as whether current mesh network has a root.

mesh_event_find_network_t find_network
  network found that can join

mesh_event_router_switch_t router_switch
  new router information

mesh_event_ps_duty_t ps_duty
  PS duty information

union mesh_rc_config_t
  #include <esp_mesh.h> Vote address configuration.

Public Members

int attempts
  max vote attempts before a new root is elected automatically by mesh network. (min:15, 15 by default)
```
mesh_addr_t rc_addr
    a new root address specified by users for API esp_mesh_waive_root()

**Structures**

**struct mip_t**

IP address and port.

**Public Members**

ip4_addr_t ip4
    IP address

uint16_t port
    port

struct mesh_event_channel_switch_t
    Channel switch information.

**Public Members**

uint8_t channel
    new channel

struct mesh_event_connected_t
    Parent connected information.

**Public Members**

wifi_event_sta_connected_t connected
    parent information, same as Wi-Fi event SYSTEM_EVENT_STA_CONNECTED does

uint16_t self_layer
    layer

uint8_t duty
    parent duty

struct mesh_event_no_parent_found_t
    No parent found information.

**Public Members**

int scan_times
    scan times being through
struct `mesh_event_layer_change_t`
Layer change information.

**Public Members**

`uint16_t new_layer`
new layer

struct `mesh_event_vote_started_t`
vote started information

**Public Members**

`int reason`
vote reason, vote could be initiated by children or by the root itself

`int attempts`
max vote attempts before stopped

`mesh_addr_t rc_addr`
root address specified by users via API `esp_mesh_waive_root()`

struct `mesh_event_find_network_t`
find a mesh network that this device can join

**Public Members**

`uint8_t channel`
channel number of the new found network

`uint8_t router_bssid[6]`
router BSSID

struct `mesh_event_root_switch_req_t`
Root switch request information.

**Public Members**

`int reason`
root switch reason, generally root switch is initialized by users via API `esp_mesh_waive_root()`

`mesh_addr_t rc_addr`
the address of root switch requestor

struct `mesh_event_root_conflict_t`
Other powerful root address.
Public Members

int8_t rssi
rssi with router

uint16_t capacity
the number of devices in current network

uint8_t addr[6]
other powerful root address

struct mesh_event_routing_table_change_t
Routing table change.

Public Members

uint16_t rt_size_new
the new value

uint16_t rt_size_change
the changed value

struct mesh_event_root_fixed_t
Root fixed.

Public Members

bool is_fixed
status

struct mesh_event_scan_done_t
Scan done event information.

Public Members

uint8_t number
the number of APs scanned

struct mesh_event_network_state_t
Network state information.

Public Members

bool is_rootless
whether current mesh network has a root
struct mesh_event_ps_duty_t
    PS duty information.

    Public Members

    uint8_t_t duty
        parent or child duty

    mesh_event_child_connected_t child_connected
        child info

struct mesh_opt_t
    Mesh option.

    Public Members

    uint8_t_t type
        option type

    uint16_t_t len
        option length

    uint8_t_t* val
        option value

struct mesh_data_t
    Mesh data for esp_mesh_send() and esp_mesh_recv()

    Public Members

    uint8_t_t* data
        data

    uint16_t_t size
        data size

    mesh_proto_t proto
        data protocol

    mesh_qos_t qos
        data type of service

struct mesh_router_t
    Router configuration.
Chapter 2. API

Public Members

```c
uint8_t ssid[32]
    SSID

uint8_t ssid_len
    length of SSID

uint8_t bssid[6]
    BSSID, if this value is specified, users should also specify “allow_router_switch”.

uint8_t password[64]
    password
```

bool allow_router_switch
if the BSSID is specified and this value is also set, when the router of this specified BSSID fails to be found after “fail” (mesh_attempts_t) times, the whole network is allowed to switch to another router with the same SSID. The new router might also be on a different channel. The default value is false. There is a risk that if the password is different between the new switched router and the previous one, the mesh network could be established but the root will never connect to the new switched router.

```c
struct mesh_ap_cfg_t
    Mesh softAP configuration.
```

Public Members

```c
uint8_t password[64]
    mesh softAP password

uint8_t max_connection
    max number of stations allowed to connect in, default 6, max 10 = max_connection + non-mesh_max_connection max mesh connections

uint8_t nonmesh_max_connection
    max non-mesh connections
```

```c
struct mesh_cfg_t
    Mesh initialization configuration.
```

Public Members

```c
uint8_t channel
    channel, the mesh network on
```

bool allow_channel_switch
if this value is set, when “fail” (mesh_attempts_t) times is reached, device will change to a full channel scan for a network that could join. The default value is false.
mesh_addr_t mesh_id
mesh network identification

mesh_router_t router
router configuration

mesh_ap_cfg_t mesh_ap
mesh softAP configuration

const mesh_crypto_funcs_t *crypto_funcs
crypto functions

struct mesh_vote_t
Vote.

Public Members

float percentage
vote percentage threshold for approval of being a root

bool is_rc_specified
if true, rc_addr shall be specified (Unimplemented). if false, attempts value shall be specified to make
network start root election.

mesh_rc_config_t config
vote address configuration

struct mesh_tx_pending_t
The number of packets pending in the queue waiting to be sent by the mesh stack.

Public Members

int to_parent
to parent queue

int to_parent_p2p
to parent (P2P) queue

int to_child
to child queue

int to_child_p2p
to child (P2P) queue

int mgmt
management queue
Chapter 2. API

int broadcast
    broadcast and multicast queue

struct mesh_rx_pending_t
    The number of packets available in the queue waiting to be received by applications.

Public Members

int toDS
    to external DS

int toSelf
    to self

Macros

MESH_ROOT_LAYER
    root layer value

MESH_MTU
    max transmit unit(in bytes)

MESH_MPS
    max payload size(in bytes)

ESP_ERR_MESH_WIFI_NOT_START
    Mesh error code definition.
    Wi-Fi isn’t started

ESP_ERR_MESH_NOT_INIT
    mesh isn’t initialized

ESP_ERR_MESH_NOT_CONFIG
    mesh isn’t configured

ESP_ERR_MESH_NOT_START
    mesh isn’t started

ESP_ERR_MESH_NOT_SUPPORT
    not supported yet

ESP_ERR_MESH_NOT_ALLOWED
    operation is not allowed

ESP_ERR_MESH_NO_MEMORY
    out of memory
ESP_ERR_MESH_ARGUMENT
    illegal argument

ESP_ERR_MESH_EXCEED_MTU
    packet size exceeds MTU

ESP_ERR_MESH_TIMEOUT
    timeout

ESP_ERR_MESH_DISCONNECTED
    disconnected with parent on station interface

ESP_ERR_MESH_QUEUE_FAIL
    queue fail

ESP_ERR_MESH_QUEUE_FULL
    queue full

ESP_ERR_MESH_NO_PARENT_FOUND
    no parent found to join the mesh network

ESP_ERR_MESH_NO_ROUTE_FOUND
    no route found to forward the packet

ESP_ERR_MESH_OPTION_NULL
    no option found

ESP_ERR_MESH_OPTION_UNKNOWN
    unknown option

ESP_ERR_MESH_XON_NO_WINDOW
    no window for software flow control on upstream

ESP_ERR_MESH_INTERFACE
    low-level Wi-Fi interface error

ESP_ERR_MESH_DISCARD_DUPLICATE
    discard the packet due to the duplicate sequence number

ESP_ERR_MESH_DISCARD
    discard the packet

ESP_ERR_MESH_VOTING
    vote in progress

ESP_ERR_MESH_XMIT
    XMIT
**Chapter 2. API 参考**

**ESP_ERR_MESH_QUEUE_READ**
error in reading queue

**ESP_ERR_MESH_PS**
mesh PS is not specified as enable or disable

**ESP_ERR_MESH_RECV_RELEASE**
release esp_mesh_recv_toDS

**MESH_DATA_ENC**
Flags bitmap for esp_mesh_send() and esp_mesh_recv()
data encrypted (Unimplemented)

**MESH_DATA_P2P**
point-to-point delivery over the mesh network

**MESH_DATA_FROMDS**
receive from external IP network

**MESH_DATA_TODS**
identify this packet is target to external IP network

**MESH_DATA_NONBLOCK**
esp_mesh_send() non-block

**MESH_DATA_DROP**
in the situation of the root having been changed, identify this packet can be dropped by new root

**MESH_DATA_GROUP**
identify this packet is target to a group address

**MESH_OPT_SEND_GROUP**
Option definitions for esp_mesh_send() and esp_mesh_recv()
data transmission by group; used with esp_mesh_send() and shall have payload

**MESH_OPT_RECV_DS_ADDR**
return a remote IP address; used with esp_mesh_send() and esp_mesh_recv()

**MESH_ASSOC_FLAG_VOTE_IN_PROGRESS**
Flag of mesh networking IE.
vote in progress

**MESH_ASSOC_FLAG_NETWORK_FREE**
no root in current network

**MESH_ASSOC_FLAG_ROOTS_FOUND**
root conflict is found
**MESH_ASSOC_FLAG_ROOT_FIXED**

fixed root

**MESH_PS_DEVICE_DUTY_REQUEST**

Mesh PS (Power Save) duty cycle type.

requests to join a network PS without specifying a device duty cycle. After the device joins the network, a network duty cycle will be provided by the network

**MESH_PS_DEVICE_DUTY_DEMAND**

requests to join a network PS and specifies a demanded device duty cycle

**MESH_PS_NETWORK_DUTY_MASTER**

indicates the device is the NWK-DUTY-MASTER (network duty cycle master)

**MESH_PS_NETWORK_DUTY_APPLIED_ENTIRE**

Mesh PS (Power Save) duty cycle applied rule.

**MESH_PS_NETWORK_DUTY_APPLIED_UPLINK**

**MESH_INIT_CONFIG_DEFAULT**

### Type Definitions

typedef `mesh_addr_t` mesh_event_root_address_t

Root address.

typedef `wifi_event_sta_disconnected_t` mesh_event_disconnected_t

Parent disconnected information.

typedef `wifi_event_ap_staconnected_t` mesh_event_child_connected_t

Child connected information.

typedef `wifi_event_ap_stadisconnected_t` mesh_event_child_disconnected_t

Child disconnected information.

typedef `wifi_event_sta_connected_t` mesh_event_router_switch_t

New router information.

### Enumerations

enum mesh_event_id_t

Enumerated list of mesh event id.

Values:

enumerator **MESH_EVENT_STARTED**

mesh is started

enumerator **MESH_EVENT_STOPPED**

mesh is stopped
enumerator `MESH_EVENT_CHANNEL_SWITCH` 
channel switch

enumerator `MESH_EVENT_CHILD_CONNECTED` 
a child is connected on softAP interface

enumerator `MESH_EVENT_CHILD_DISCONNECTED` 
a child is disconnected on softAP interface

enumerator `MESH_EVENT_ROUTING_TABLE_ADD` 
routing table is changed by adding newly joined children

enumerator `MESH_EVENT_ROUTING_TABLE_REMOVE` 
routing table is changed by removing leave children

enumerator `MESH_EVENT_PARENT_CONNECTED` 
parent is connected on station interface

enumerator `MESH_EVENT_PARENT_DISCONNECTED` 
parent is disconnected on station interface

enumerator `MESH_EVENT_NO_PARENT_FOUND` 
no parent found

enumerator `MESH_EVENT_LAYER_CHANGE` 
layer changes over the mesh network

enumerator `MESH_EVENT_TODS_STATE` 
state represents whether the root is able to access external IP network

enumerator `MESH_EVENT_VOTE_STARTED` 
the process of voting a new root is started either by children or by the root

enumerator `MESH_EVENT_VOTE_STOPPED` 
the process of voting a new root is stopped

enumerator `MESH_EVENT_ROOT_ADDRESS` 
the root address is obtained. It is posted by mesh stack automatically.

enumerator `MESH_EVENT_ROOT_SWITCH_REQ` 
root switch request sent from a new voted root candidate

enumerator `MESH_EVENT_ROOT_SWITCH_ACK` 
root switch acknowledgment responds the above request sent from current root

enumerator `MESH_EVENT_ROOT_ASKED_YIELD` 
the root is asked yield by a more powerful existing root. If self organized is disabled and this device is specified to be a root by users, users should set a new parent for this device. If self organized is enabled, this device will find a new parent by itself, users could ignore this event.
Chapter 2. API

enumerator MESH_EVENT_ROOT_FIXED
    when devices join a network, if the setting of Fixed Root for one device is different from that of its parent, the device will update the setting the same as its parent’s. Fixed Root Setting of each device is variable as that setting changes of the root.

enumerator MESH_EVENT_SCAN_DONE
    if self-organized networking is disabled, user can call esp_wifi_scan_start() to trigger this event, and add the corresponding scan done handler in this event.

enumerator MESH_EVENT_NETWORK_STATE
    network state, such as whether current mesh network has a root.

enumerator MESH_EVENT_STOP_RECONNECTION
    the root stops reconnecting to the router and non-root devices stop reconnecting to their parents.

enumerator MESH_EVENT_FIND_NETWORK
    when the channel field in mesh configuration is set to zero, mesh stack will perform a full channel scan to find a mesh network that can join, and return the channel value after finding it.

enumerator MESH_EVENT_ROUTER_SWITCH
    if users specify BSSID of the router in mesh configuration, when the root connects to another router with the same SSID, this event will be posted and the new router information is attached.

enumerator MESH_EVENT_PS_PARENT_DUTY
    parent duty

enumerator MESH_EVENT_PS_CHILD_DUTY
    child duty

enumerator MESH_EVENT_PS_DEVICE_DUTY
    device duty

enumerator MESH_EVENT_MAX

enum mesh_type_t
    Device type.
    Values:

enumerator MESH_IDLE
    hasn’t joined the mesh network yet

enumerator MESH_ROOT
    the only sink of the mesh network. Has the ability to access external IP network

enumerator MESH_NODE
    intermediate device. Has the ability to forward packets over the mesh network

enumerator MESH_LEAF
    has no forwarding ability
enumerator **MESH_STA**
   connect to router with a standalone Wi-Fi station mode, no network expansion capability

enum **mesh_proto_t**
   Protocol of transmitted application data.
   *Values:*
   
   enumerator **MESH_PROTO_BIN**
      binary
   
   enumerator **MESH_PROTO_HTTP**
      HTTP protocol
   
   enumerator **MESH_PROTO_JSON**
      JSON format
   
   enumerator **MESH_PROTO_MQTT**
      MQTT protocol
   
   enumerator **MESH_PROTO_AP**
      IP network mesh communication of node’s AP interface
   
   enumerator **MESH_PROTO_STA**
      IP network mesh communication of node’s STA interface

enum **mesh_tos_t**
   For reliable transmission, mesh stack provides three type of services.
   *Values:*
   
   enumerator **MESH_TOS_P2P**
      provide P2P (point-to-point) retransmission on mesh stack by default
   
   enumerator **MESH_TOS_E2E**
      provide E2E (end-to-end) retransmission on mesh stack (Unimplemented)
   
   enumerator **MESH_TOS_DEF**
      no retransmission on mesh stack

enum **mesh_vote_reason_t**
   Vote reason.
   *Values:*
   
   enumerator **MESH_VOTE_REASON_ROOT_INITIATED**
      vote is initiated by the root
   
   enumerator **MESH_VOTE_REASON_CHILD_INITIATED**
      vote is initiated by children
enum mesh_disconnect_reason_t
   Mesh disconnect reason code.
   Values:

   enumerator MESH_REASON_CYCLIC
cyclic is detected

   enumerator MESH_REASON_PARENT_IDLE
parent is idle

   enumerator MESH_REASON_LEAF
the connected device is changed to a leaf

   enumerator MESH_REASON_DIFF_ID
in different mesh ID

   enumerator MESH_REASON_ROOTS
root conflict is detected

   enumerator MESH_REASON_PARENT_STOPPED
parent has stopped the mesh

   enumerator MESH_REASON_SCAN_FAIL
scan fail

   enumerator MESH_REASON_IE_UNKNOWN
unknown IE

   enumerator MESH_REASON_WAIVE_ROOT
waive root

   enumerator MESH_REASON_PARENT_WORSE
parent with very poor RSSI

   enumerator MESH_REASON_EMPTY_PASSWORD
use an empty password to connect to an encrypted parent

   enumerator MESH_REASON_PARENT_UNENCRYPTED
connect to an unencrypted parent/router

enum esp_mesh_topology_t
   Mesh topology.
   Values:

   enumerator MESH_TOPO_TREE
tree topology
enumerator MESH_TOPO_CHAIN
chain topology

denum mesh_event_toDS_state_t

The reachability of the root to a DS (distribute system)

Values:

enumerator MESH_TODS_UNREACHABLE
the root isn’t able to access external IP network

enumerator MESH_TODS_REACHABLE
the root is able to access external IP network

SmartConfig

The SmartConfig™ is a provisioning technology developed by TI to connect a new Wi-Fi device to a Wi-Fi network. It uses a mobile app to broadcast the network credentials from a smartphone, or a tablet, to an un-provisioned Wi-Fi device.

The advantage of this technology is that the device does not need to directly know SSID or password of an Access Point (AP). This information is provided using the smartphone. This is particularly important to headless device and systems, due to their lack of a user interface.

If you are looking for other options to provision your ESP32 devices, check API.

Application Example Connect ESP32 to target AP using SmartConfig: wifi/smarty_config.

API Reference

Header File

• components/esp_wifi/include/esp_smartconfig.h

Functions

const char *esp_smartconfig_get_version (void)
Get the version of SmartConfig.

返倠
• SmartConfig version const char.

esp_err_t esp_smartconfig_start (const smartconfig_start_config_t *config)
Start SmartConfig, config ESP device to connect AP. You need to broadcast information by phone APP. Device sniffer special packets from the air that containing SSID and password of target AP.

Attention 1. This API can be called in station or softAP-station mode.
Attention 2. Can not call esp_smartconfig_start twice before it finish, please call esp_smartconfig_stop first.

参数 config - pointer to smartconfig start configure structure
返回
• ESP_OK: succeed
• others: fail
**esp_err_t** **esp_smartconfig_stop**(void)

Stop SmartConfig, free the buffer taken by esp_smartconfig_start.

**Attention** Whether connect to AP succeed or not, this API should be called to free memory taken by smartconfig_start.

返回
• ESP_OK: succeed
• others: fail

**esp_err_t** **esp_esptouch_set_timeout**(uint8_t time_s)

Set timeout of SmartConfig process.

**Attention** Timing starts from SC_STATUS_FIND_CHANNEL status. SmartConfig will restart if timeout.

参数 time_s – range 15s~255s, offset:45s.

返回
• ESP_OK: succeed
• others: fail

**esp_err_t** **esp_smartconfig_set_type**(smartconfig_type_t type)

Set protocol type of SmartConfig.

**Attention** If users need to set the SmartConfig type, please set it before calling esp_smartconfig_start.

参数 type – Choose from the smartconfig_type_t.

返回
• ESP_OK: succeed
• others: fail

**esp_err_t** **esp_smartconfig_fast_mode**(bool enable)

Set mode of SmartConfig. default normal mode.

**Attention**
1. Please call it before API esp_smartconfig_start.
2. Fast mode have corresponding APP(phone).
3. Two mode is compatible.

参数 enable – false-disable(default); true-enable;

返回
• ESP_OK: succeed
• others: fail

**esp_err_t** **esp_smartconfig_get_rvd_data**(uint8_t *rvd_data, uint8_t len)

Get reserved data of ESPTouch v2.

参数
• rvd_data – reserved data
• len – length of reserved data

返回
• ESP_OK: succeed
• others: fail
Structures

struct `smartconfig_event_got_ssid_pswd_t`
Argument structure for SC_EVENT_GOT_SSID_PSWD event

Public Members

`uint8_t ssid[32]`
SSID of the AP. Null terminated string.

`uint8_t password[64]`
Password of the AP. Null terminated string.

`bool bssid_set`
whether set MAC address of target AP or not.

`uint8_t bssid[6]`
MAC address of target AP.

`sxartconfig_type_t type`
Type of smartconfig(ESPTouch or AirKiss).

`uint8_t token`
Token from cellphone which is used to send ACK to cellphone.

`uint8_t cellphone_ip[4]`
IP address of cellphone.

struct `smartconfig_start_config_t`
Configure structure for esp_smartconfig_start

Public Members

`bool enable_log`
Enable smartconfig logs.

`bool esp_touch_v2_enable_crypt`
Enable ESPTouch v2 crypt.

`char *esp_touch_v2_key`
ESPTouch v2 crypt key, len should be 16.

Macros

`SMARTCONFIG_START_CONFIG_DEFAULT()`
Enumerations

enum smartconfig_type_t

Values:

- SC_TYPE_ESPTOUCH
  protocol: ESPTouch

- SC_TYPE_AIRKISS
  protocol: AirKiss

- SC_TYPE_ESPTOUCH_AIRKISS
  protocol: ESPTouch and AirKiss

- SC_TYPE_ESPTOUCH_V2
  protocol: ESPTouch v2

doc Section 2. API 参考

Wi-Fi 库

概述 Wi-Fi 库支持配置及监控 ESP32 Wi-Fi 连网功能。支持配置：
- station 模式 (即 STA 模式或 Wi-Fi 客户端模式)，此时 ESP32 连接到接入点 (AP)。
- AP 模式 (即 Soft-AP 模式或接入点模式)，此时基站连接到 ESP32。
- station/AP 共存模式 (ESP32 既是接入点，同时又作为基站连接到另外 一个接入点)。
- 上述模式的各种安全模式 (WPA、WPA2 及 WEP 等)。
- 扫描接入点 (包括主动扫描及被动扫描)。
- 使用混杂模式监控 IEEE802.11 Wi-Fi 数据包。

应用示例 ESP-IDF 仓库的 wifi 目录下提供了演示 Wi-Fi 库功能的几个应用示例，请查看 README 了解更多详细信息。

API 参考

Espressif Systems 833 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
Chapter 2. API参考

Header File
- components/esp_wifi/include/esp_wifi.h

Functions

**esp_err_t esp_wifi_init** (const wifi_init_config_t *config)
- Initialize WiFi Allocate resource for WiFi driver, such as WiFi control structure, RX/TX buffer, WiFi NVS structure etc. This WiFi also starts WiFi task.

**Attention** 1. This API must be called before all other WiFi API can be called
- 2. Always use WIFI_INIT_CONFIG_DEFAULT macro to initialize the configuration to default values, this can guarantee all the fields get correct value when more fields are added into wifi_init_config_t in future release. If you want to set your own initial values, overwrite the default values which are set by WIFI_INIT_CONFIG_DEFAULT. Please be notified that the field ‘magic’ of wifi_init_config_t should always be WIFI_INIT_CONFIG_MAGIC!

参数 config - pointer to WiFi initialized configuration structure; can point to a temporary variable.
- ESP_OK: succeed
- ESP_ERR_NO_MEM: out of memory
- others: refer to error code esp_err.h

**esp_err_t esp_wifi_deinit** (void)
- Deinit WiFi Free all resource allocated in esp_wifi_init and stop WiFi task.

**Attention** 1. This API should be called if you want to remove WiFi driver from the system

参数 config - WiFi operating mode
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

**Set the WiFi operating mode as station, soft-AP or station+soft-AP, The default mode is station mode.**

**esp_err_t esp_wifi_set_mode** (wifi_mode_t mode)
- Set the WiFi operating mode.

参数 mode - WiFi operating mode
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error code in esp_err.h

**esp_err_t esp_wifi_get_mode** (wifi_mode_t *mode)
- Get current operating mode of WiFi.

参数 mode -[out] store current WiFi mode
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
### Chapter 2. API 参考

**esp_err_t esp_wifi_start (void)**

Start WiFi according to current configuration. If mode is WIFI_MODE_STA, it create station control block and start station. If mode is WIFI_MODE_AP, it create soft-AP control block and start soft-AP. If mode is WIFI_MODE_APSTA, it create soft-AP and station control block and start soft-AP and station.

- **ESP_OK**: succeed
- **ESP_ERR_WIFI_NOT_INIT**: WiFi is not initialized by esp_wifi_init
- **ESP_ERR_INVALID_ARG**: invalid argument
- **ESP_ERR_NO_MEM**: out of memory
- **ESP_ERR_WIFI_CONN**: WiFi internal error, station or soft-AP control block wrong
- **ESP_FAIL**: other WiFi internal errors

**esp_err_t esp_wifi_stop (void)**

Stop WiFi. If mode is WIFI_MODE_STA, it stop station and free station control block. If mode is WIFI_MODE_AP, it stop soft-AP and free soft-AP control block. If mode is WIFI_MODE_APSTA, it stop station/soft-AP and free station/soft-AP control block.

- **ESP_OK**: succeed
- **ESP_ERR_WIFI_NOT_INIT**: WiFi is not initialized by esp_wifi_init

**esp_err_t esp_wifi_restore (void)**

Restore WiFi stack persistent settings to default values.

This function will reset settings made using the following APIs:

- esp_wifi_set_bandwidth,
- esp_wifi_set_protocol,
- esp_wifi_set_config related
- esp_wifi_set_mode

- **ESP_OK**: succeed
- **ESP_ERR_WIFI_NOT_INIT**: WiFi is not initialized by esp_wifi_init

**esp_err_t esp_wifi_connect (void)**

Connect the ESP32 WiFi station to the AP.

**Attention** 1. This API only impact WIFI_MODE_STA or WIFI_MODE_APSTA mode

**Attention** 2. If the ESP32 is connected to an AP, call esp_wifi_disconnect to disconnect.

**Attention** 3. The scanning triggered by esp_wifi_scan_start() will not be effective until connection between ESP32 and the AP is established. If ESP32 is scanning and connecting at the same time, ESP32 will abort scanning and return a warning message and error number ESP_ERR_WIFI_STATE. If you want to do reconnection after ESP32 received disconnect event, remember to add the maximum retry time, otherwise the called scan will not work. This is especially true when the AP doesn’t exist, and you still try reconnection after ESP32 received disconnect event with the reason code WIFI_REASON_NO_AP_FOUND.

- **ESP_OK**: succeed
- **ESP_ERR_WIFI_NOT_INIT**: WiFi is not initialized by esp_wifi_init
- **ESP_ERR_WIFI_NOT_STARTED**: WiFi is not started by esp_wifi_start
- **ESP_ERR_WIFI_CONN**: WiFi internal error, station or soft-AP control block wrong
- **ESP_ERR_WIFI_SSID**: SSID of AP which station connects is invalid

**esp_err_t esp_wifi_disconnect (void)**

Disconnect the ESP32 WiFi station from the AP.

- **ESP_OK**: succeed
Chapter 2. API 参考

• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi was not initialized by esp_wifi_init
• ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
• ESP_FAIL: other WiFi internal errors

`esp_err_t esp_wifi_clear_fast_connect (void)`
Currently this API is just an stub API.

返回
• ESP_OK: succeed
• others: fail

`esp_err_t esp_wifi_deauth_sta (uint16_t aid)`
deauthenticate all stations or associated id equals to aid

参数aid—when aid is 0, deauthenticate all stations, otherwise deauthenticate station whose associated id is aid

返回
• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
• ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
• ESP_ERR_INVALID_ARG: invalid argument
• ESP_ERR_WIFI_MODE: WiFi mode is wrong

`esp_err_t esp_wifi_scan_start (const wifi_scan_config_t *config, bool block)`
Scan all available APs.

Attention If this API is called, the found APs are stored in WiFi driver dynamic allocated memory and the will be freed in esp_wifi_scan_get_ap_records, so generally, call esp_wifi_scan_get_ap_records to cause the memory to be freed once the scan is done

Attention The values of maximum active scan time and passive scan time per channel are limited to 1500 milliseconds. Values above 1500ms may cause station to disconnect from AP and are not recommended.

参数
• config—configuration of scanning
• block—if block is true, this API will block the caller until the scan is done, otherwise it will return immediately

返回
• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
• ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
• ESP_ERR_WIFI_TIMEOUT: blocking scan is timeout
• ESP_ERR_WIFI_STATE: wifi still connecting when invoke esp_wifi_scan_start
• others: refer to error code in esp_err.h

`esp_err_t esp_wifi_scan_stop (void)`
Stop the scan in process.

返回
• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
• ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start

`esp_err_t esp_wifi_scan_get_ap_num (uint16_t *number)`
Get number of APs found in last scan.

Attention This API can only be called when the scan is completed, otherwise it may get wrong value.

参数 number—[out] store number of APIs found in last scan
**Chapter 2. API 参考**

### esp_err_t esp_wifi_scan_get_ap_records (uint16_t* number, wifi_ap_record_t *ap_records)

Get AP list found in last scan.

#### Parameters
- **number** - [inout] As input param, it stores max AP number ap_records can hold. As output param, it receives the actual AP number this API returns.
- **ap_records** - wifi_ap_record_t array to hold the found APs

#### Return
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NO_MEM: out of memory

### esp_err_t esp_wifi_clear_ap_list (void)

Clear AP list found in last scan.

**Attention** When the obtained ap list fails, bss info must be cleared, otherwise it may cause memory leakage.

#### Return
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- ESP_ERR_INVALID_ARG: invalid argument

### esp_err_t esp_wifi_sta_get_ap_info (wifi_ap_record_t *ap_info)

Get information of AP which the ESP32 station is associated with.

**Attention** When the obtained country information is empty, it means that the AP does not carry country information.

#### Parameter
- **ap_info** - the wifi_ap_record_t to hold AP information sta can get the connected ap’s phy mode info through the struct member phy_11b, phy_11g, phy_11n, phy_lr in the wifi_ap_record_t struct. For example, phy_11b = 1 imply that ap support 802.11b mode

#### Return
- ESP_OK: succeed
- ESP_ERR_WIFI_CONN: The station interface don’t initialized
- ESP_ERR_WIFI_NOT_CONNECT: The station is in disconnect status

### esp_err_t esp_wifi_set_ps (wifi_ps_type_t type)

Set current WiFi power save type.

**Attention** Default power save type is WIFI_PS_MIN_MODEM.

#### Parameters
- **type** - power save type

#### Return
- ESP_OK: succeed
**esp_err_t esp_wifi_get_ps(wifi_ps_type_t *type)**

Get current WiFi power save type.

**Attention** Default power save type is WIFI_PS_MIN_MODEM.

参数 type - [out] store current power save type

返回 ESP_OK: succeed

**esp_err_t esp_wifi_set_protocol(wifi_interface_t ifx, uint8_t protocol_bitmap)**

Set protocol type of specified interface. The default protocol is WiFi protocol bitmap.

**Attention** Support 802.11b or 802.11bg or 802.11bgn or LR mode

参数
- ifx - interfaces
- protocol_bitmap - WiFi protocol bitmap

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error codes in esp_err.h

**esp_err_t esp_wifi_get_protocol(wifi_interface_t ifx, uint8_t *protocol_bitmap)**

Get the current protocol bitmap of the specified interface.

参数
- ifx - interface
- protocol_bitmap - [out] store current WiFi protocol bitmap of interface ifx

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error codes in esp_err.h

**esp_err_t esp_wifi_set_bandwidth(wifi_interface_t ifx, wifi_bandwidth_t bw)**

Set the bandwidth of ESP32 specified interface.

**Attention** 1. API return false if try to configure an interface that is not enabled

2. WIFI_BW_HT40 is supported only when the interface support 11N

参数
- ifx - interface to be configured
- bw - bandwidth

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error codes in esp_err.h

**esp_err_t esp_wifi_get_bandwidth(wifi_interface_t ifx, wifi_bandwidth_t *bw)**

Get the bandwidth of ESP32 specified interface.
Chapter 2. API 

**Attention** 1. API return false if try to get a interface that is not enable

### Parameters
- **ifx** - interface to be configured
- **bw** - [out] store bandwidth of interface ifx

### Return
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument

```c
esp_err_t esp_wifi_set_channel(uint8_t primary, wifi_second_chan_t second)
```
Set primary/secondary channel of ESP32.

**Attention** 2. This API should be called after esp_wifi_start()

**Attention** 3. When ESP32 is in softAP mode, this API should not be called when softAP has connected to external STAs

**Attention** 4. When ESP32 is in STA+softAP mode, this API should not be called when in the scenarios described above

**Attention** 5. The channel info set by this API will not be stored in NVS. So If you want to remerber the channel used before wifi stop, you need to call this API again after wifi start, or you can call esp_wifi_set_config() to store the channel info in NVS.

### Parameters
- **primary** - for HT20, primary is the channel number, for HT40, primary is the primary channel
- **second** - for HT20, second is ignored, for HT40, second is the second channel

### Return
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument

```c
esp_err_t esp_wifi_get_channel(uint8_t* primary, wifi_second_chan_t* second)
```
Get the primary/secondary channel of ESP32.

**Attention** 1. API return false if try to get a interface that is not enable

### Parameters
- **primary** - store current primary channel
- **second** - [out] store current second channel

### Return
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

```c
esp_err_t esp_wifi_set_country(const wifi_country_t* country)
```
Configure country info

**Attention** 1. It is discouraged to call this API since this doesn’t validate the per-country rules, it’s up to the user to fill in all fields according to local regulations. Please use esp_wifi_set_country_code instead.
**Attention** 2. The default country is “01” (world safe mode) {.cc=’01’, .schan=1, .nchan=11, .policy=WIFI_COUNTRY_POLICY_AUTO}.

**Attention** 3. The third octect of country code string is one of the following: ‘ ’, ‘O’, ‘1’, ‘X’, otherwise it is considered as ‘ ‘.

**Attention** 4. When the country policy is WIFI_COUNTRY_POLICY_AUTO, the country info of the AP to which the station is connected is used. E.g. if the configured country info is {.cc=”US”, .schan=1, .nchan=11} and the country info of the AP to which the station is connected is {.cc=”JP”, .schan=1, .nchan=14} then the country info that will be used is {.cc=”JP”, .schan=1, .nchan=14}. If the station disconnected from the AP the country info is set back to the country info of the station automatically, {.cc=”US”, .schan=1, .nchan=11} in the example.

**Attention** 5. When the country policy is WIFI_COUNTRY_POLICY_MANUAL, then the configured country info is used always.

**Attention** 6. When the country info is changed because of configuration or because the station connects to a different external AP, the country IE in probe response/beacon of the soft-AP is also changed.

**Attention** 7. The country configuration is stored into flash.

**Attention** 8. When this API is called, the PHY init data will switch to the PHY init data type corresponding to the country info.

```
country – the configured country info
```

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

```c
esp_err_t esp_wifi_get_country (wifi_country_t *country)
```

get the current country info

```
country – country info
```

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

```c
esp_err_t esp_wifi_set_mac (wifi_interface_t ifx, const uint8_t mac[6])
```

Set MAC address of the ESP32 WiFi station or the soft-AP interface.

**Attention** 1. This API can only be called when the interface is disabled.

**Attention** 2. ESP32 soft-AP and station have different MAC addresses, do not set them to be the same.

**Attention** 3. The bit 0 of the first byte of ESP32 MAC address can not be 1. For example, the MAC address can set to be “1a:XX:XX:XX:XX:XX”, but can not be “15:XX:XX:XX:XX:XX”.

```
ifix – interface
```
```
mac – the MAC address
```

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_WIFI_MAC: invalid mac address
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- others: refer to error codes in esp_err.h

```c
esp_err_t esp_wifi_get_mac (wifi_interface_t ifx, uint8_t mac[6])
```

Get mac of specified interface.

```
ifix – interface
```
Chapter 2. API 参考

- **mac** [out] store mac of the interface ifx

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface

`esp_err_t esp_wifi_set_promiscuous_rx_cb (wifi_promiscuous_cb_t cb)`

Register the RX callback function in the promiscuous mode.

Each time a packet is received, the registered callback function will be called.

参数 **cb** - callback

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

`esp_err_t esp_wifi_set_promiscuous (bool en)`

Enable the promiscuous mode.

参数 **en** - false - disable, true - enable

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

`esp_err_t esp_wifi_get_promiscuous (bool *en)`

Get the promiscuous mode.

参数 **en** [out] store the current status of promiscuous mode

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

`esp_err_t esp_wifi_set_promiscuous_filter (const wifi_promiscuous_filter_t *filter)`

Enable the promiscuous mode packet type filter.

备注: The default filter is to filter all packets except WIFI_PKT_MISC

参数 **filter** - the packet type filtered in promiscuous mode.

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

`esp_err_t esp_wifi_get_promiscuous_filter (wifi_promiscuous_filter_t *filter)`

Get the promiscuous filter.

参数 **filter** [out] store the current status of promiscuous filter

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

`esp_err_t esp_wifi_set_promiscuous_ctrl_filter (const wifi_promiscuous_filter_t *filter)`

Enable subtype filter of the control packet in promiscuous mode.

备注: The default filter is to filter none control packet.

参数 **filter** - the subtype of the control packet filtered in promiscuous mode.
Chapter 2. API Reference

返回
• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

`esp_err_t esp_wifi_get_promiscuous_ctrl_filter (wifi_promiscuous_filter_t *filter)`
Get the subtype filter of the control packet in promiscuous mode.

参数 **filter** - [out] store the current status of subtype filter of the control packet in promiscuous mode

返回
• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
• ESP_ERR_WIFI_ARG: invalid argument

`esp_err_t esp_wifi_set_config (wifi_interface_t interface, wifi_config_t *conf)`
Set the configuration of the ESP32 STA or AP.

**Attention** 1. This API can be called only when specified interface is enabled, otherwise, API fail
**Attention** 2. For station configuration, bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.
**Attention** 3. ESP32 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP32 station.
**Attention** 4. The configuration will be stored in NVS

参数
• **interface** – interface
• **conf** – station or soft-AP configuration

返回
• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
• ESP_ERR_INVALID_ARG: invalid argument
• ESP_ERR_WIFI_IF: invalid interface
• ESP_ERR_WIFI_MODE: invalid mode
• ESP_ERR_WIFI_PASSWORD: invalid password
• ESP_ERR_WIFI_NVS: WiFi internal NVS error
• others: refer to the erro code in esp_err.h

`esp_err_t esp_wifi_get_config (wifi_interface_t interface, wifi_config_t *conf)`
Get configuration of specified interface.

参数
• **interface** – interface
• **conf** – [out] station or soft-AP configuration

返回
• ESP_OK: succeed
• ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
• ESP_ERR_INVALID_ARG: invalid argument
• ESP_ERR_WIFI_IF: invalid interface

`esp_err_t esp_wifi_ap_get_sta_list (wifi_sta_list_t *sta)`
Get STAs associated with soft-AP.

**Attention** SSC only API

参数 **sta** – [out] station list ap can get the connected sta’s phy mode info through the struct member phy_11b, phy_11g, phy_11n, phy_lr in the wifi_sta_info_t struct. For example, phy_11b = 1 imply that sta support 802.11b mode
Chapter 2. API

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- ESP_ERR_WIFI_CONN: WiFi internal error, the station/soft-AP control block is invalid

`esp_err_t esp_wifi_ap_get_sta_aid(const uint8_t mac[6], uint16_t *aid)`
Get AID of STA connected with soft-AP.

参数
- `mac` - STA’s mac address
- `aid` - [out] Store the AID corresponding to STA mac

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NOT_FOUND: Requested resource not found
- ESP_ERR_WIFI_CONN: WiFi internal error, the station/soft-AP control block is invalid

`esp_err_t esp_wifi_set_storage(wifi_storage_t storage)`
Set the WiFi API configuration storage type.

Attention 1. The default value is WIFI_STORAGE_FLASH

参数 `storage` - storage type

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

`esp_err_t esp_wifi_set_vendor_ie(bool enable, wifi_vendor_ie_type_t type, wifi_vendor_ie_id_t idx, const void *vnd_ie)`
Set 802.11 Vendor-Specific Information Element.

参数
- `enable` - If true, specified IE is enabled. If false, specified IE is removed.
- `type` - Information Element type. Determines the frame type to associate with the IE.
- `idx` - Index to set or clear. Each IE type can be associated with up to two elements (indices 0 & 1).
- `vnd_ie` - Pointer to vendor specific element data. First 6 bytes should be a header with fields matching `vendor_ie_data_t`. If enable is false, this argument is ignored and can be NULL. Data does not need to remain valid after the function returns.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init()
- ESP_ERR_INVALID_ARG: Invalid argument, including if first byte of vnd_ie is not WIFI_VENDOR_IE_ELEMENT_ID (0xDD) or second byte is an invalid length.
- ESP_ERR_NO_MEM: Out of memory

`esp_err_t esp_wifi_set_vendor_ie_cb(esp_vendor_ie_cb_t cb, void *ctx)`
Register Vendor-Specific Information Element monitoring callback.

参数
- `cb` - Callback function
- `ctx` - Context argument, passed to callback function.

Return

- ESP_OK: succeed
**ESP_ERR_WIFI_NOT_INIT**: WiFi is not initialized by esp_wifi_init

```c
esp_err_t esp_wifi_init()
```

Set maximum transmitting power after WiFi start.

**Attention** 1. Maximum power before wifi startup is limited by PHY init data bin.
**Attention** 2. The value set by this API will be mapped to the max tx_power of the structure `wifi_country_t` variable.
**Attention** 3. Mapping Table `{Power, max_tx_power} = {{8, 2}, {20, 5}, {28, 7}, {34, 8}, {44, 11}, {52, 13}, {56, 14}, {60, 15}, {66, 16}, {72, 18}, {80, 20}}`.
**Attention** 4. Power unit is 0.25dBm, range is $[8, 84]$ corresponding to 2dBm - 20dBm.
**Attention** 5. Relationship between set value and actual value. As follows: `{set value range, actual value} = {{[8, 19], 8}, {[20, 27], 20}, {[28, 33], 28}, {[34, 43], 28}, {[44, 51], 44}, {[52, 55], 52}, {[56, 59], 56}, {[60, 65], 60}, {[66, 71], 66}, {[72, 79], 72}, {[80, 84], 80}}`.

**esp_err_t esp_wifi_set_max_tx_power** (int8_t power)

Set maximum transmitting power after WiFi start.

**参数** power – Maximum WiFi transmitting power.

**返回**

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is out of range

```c
esp_err_t esp_wifi_get_max_tx_power (int8_t *power)
```

Get maximum transmitting power after WiFi start.

**参数** power – Maximum WiFi transmitting power, unit is 0.25dBm.

**返回**

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument

```c
esp_err_t esp_wifi_set_event_mask (uint32_t mask)
```

Set mask to enable or disable some WiFi events.

**Attention** 1. Mask can be created by logical OR of various WIFI_EVENT_MASK constants. Events which have corresponding bit set in the mask will not be delivered to the system event handler.
**Attention** 2. Default WiFi event mask is WIFI_EVENT_MASK_AP_PROBEREQRECVED.
**Attention** 3. There may be lots of stations sending probe request data around. Don’t unmask this event unless you need to receive probe request data.

**参数** mask – WiFi event mask.

**返回**

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

```c
esp_err_t esp_wifi_get_event_mask (uint32_t *mask)
```

Get mask of WiFi events.

**参数** mask – WiFi event mask.

**返回**

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument
**esp_err_t esp_wifi_80211_tx** *(wifi_interface_t ifx, const void *buffer, int len, bool en_sys_seq)*

Send raw ieee80211 data.

**Attention** Currently only support for sending beacon/probe request/probe response/action and non-QoS data frame

**参数**
- **ifx** – interface if the Wi-Fi mode is Station, the ifx should be WIFI_IF_STA. If the Wi-Fi mode is SoftAP, the ifx should be WIFI_IF_AP. If the Wi-Fi mode is Station+SoftAP, the ifx should be WIFI_IF_STA or WIFI_IF_AP. If the ifx is wrong, the API returns ESP_ERR_WIFI_IF.
- **buffer** – raw ieee80211 buffer
- **len** – the length of raw buffer, the len must be <= 1500 Bytes and >= 24 Bytes
- **en_sys_seq** – indicate whether use the internal sequence number. If en_sys_seq is false, the sequence in raw buffer is unchanged, otherwise it will be overwritten by WiFi driver with the system sequence number. Generally, if esp_wifi_80211_tx is called before the Wi-Fi connection has been set up, both en_sys_seq=true and en_sys_seq=false are fine. However, if the API is called after the Wi-Fi connection has been set up, en_sys_seq must be true, otherwise ESP_ERR_WIFI_ARG is returned.

**返回**
- ESP_OK: success
- ESP_ERR_WIFI_IF: Invalid interface
- ESP_ERR_INVALID_ARG: Invalid parameter
- ESP_ERR_WIFI_NO_MEM: out of memory

**esp_err_t esp_wifi_set_csi_rx_cb** *(wifi_csi_cb_t cb, void *ctx)*

Register the RX callback function of CSI data.

Each time a CSI data is received, the callback function will be called.

**参数**
- **cb** – callback
- **ctx** – context argument, passed to callback function

**返回**
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

**esp_err_t esp_wifi_set_csi_config** *(const wifi_csi_config_t *config)*

Set CSI data configuration.

**return**
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start or promiscuous mode is not enabled
- ESP_ERR_INVALID_ARG: invalid argument

**参数**
- **config** – configuration

**esp_err_t esp_wifi_set_csi** *(bool en)*

Enable or disable CSI.
return
  • ESP_OK: succeed
  • ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  • ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start or promiscuous mode is not enabled
  • ESP_ERR_INVALID_ARG: invalid argument

参数 en - true - enable, false - disable

*esp_err_t* esp_wifi_set_ant_gpio (const wifi_ant_gpio_config_t *config)
Set antenna GPIO configuration.

参数 config - Antenna GPIO configuration.
返回
  • ESP_OK: succeed
  • ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  • ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL, invalid GPIO number etc

*esp_err_t* esp_wifi_get_ant_gpio (wifi_ant_gpio_config_t *config)
Get current antenna GPIO configuration.

参数 config - Antenna GPIO configuration.
返回
  • ESP_OK: succeed
  • ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  • ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL

*esp_err_t* esp_wifi_set_ant (const wifi_ant_config_t *config)
Set antenna configuration.

参数 config - Antenna configuration.
返回
  • ESP_OK: succeed
  • ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  • ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL, invalid antenna mode or invalid GPIO number

*esp_err_t* esp_wifi_get_ant (wifi_ant_config_t *config)
Get current antenna configuration.

参数 config - Antenna configuration.
返回
  • ESP_OK: succeed
  • ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  • ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL

*int64_t* esp_wifi_get_tsf_time (wifi_interface_t interface)
Get the TSF time in Station mode or SoftAP+Station mode if station is not connected or station doesn’t receive at least one beacon after connected, will return 0.

**Attention** Enabling power save may cause the return value inaccurate, except WiFi modem sleep

参数 interface - The interface whose tsf_time is to be retrieved.
返回 0 or the TSF time

*esp_err_t* esp_wifi_set_inactive_time (wifi_interface_t ifx, uint16_t sec)
Set the inactive time of the ESP32 STA or AP.
**Attention** 1. For Station, If the station does not receive a beacon frame from the connected SoftAP during the inactive time, disconnect from SoftAP. Default 6s.

**Attention** 2. For SoftAP, If the softAP doesn’t receive any data from the connected STA during inactive time, the softAP will force deauth the STA. Default is 300s.

**Attention** 3. The inactive time configuration is not stored into flash

- **参数**
  - *ifx* – interface to be configured.
  - *sec* – Inactive time. Unit seconds.

- **返回**
  - ESP_OK: succeed
  - ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  - ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
  - ESP_ERR_WIFI_ARG: invalid argument, For Station, if sec is less than 3. For SoftAP, if sec is less than 10.

### esp_wifi_send_inactive_time(wifi_interface_t ifx, uint16_t* sec)

Get inactive time of specified interface.

- **参数**
  - *ifx* – Interface to be configured.
  - *sec* – Inactive time. Unit seconds.

- **返回**
  - ESP_OK: succeed
  - ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  - ESP_ERR_WIFI_ARG: invalid argument

### esp_wifi_stats_dump(uint32_t modules)

Dump WiFi statistics.

- **参数**
  - *modules* – statistic modules to be dumped

- **返回**
  - ESP_OK: succeed
  - others: failed

### esp_wifi_set_rssi_threshold(int32_t rssi)

Set RSSI threshold below which APP will get an event.

- **参数**
  - *rssi* – threshold value in dbm between -100 to 0

- **返回**
  - ESP_OK: succeed
  - ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
  - ESP_ERR_WIFI_ARG: invalid argument

### esp_wifi_ftm_initiate_session(wifi_ftm_initiator_cfg_t *cfg)

Start an FTM Initiator session by sending FTM request If successful, event WIFI_EVENT_FTM_REPORT is generated with the result of the FTM procedure.

- **Attention** Use this API only in Station mode

- **参数**
  - *cfg* – FTM Initiator session configuration

- **返回**
  - ESP_OK: succeed
  - others: failed
**Chapter 2. API 参考**

```c
esp_err_t esp_wifi_ftm_end_session (void)
End the ongoing FTM Initiator session.
```

**Attention** This API works only on FTM Initiator

- ESP_OK: succeed
- others: failed

```c
esp_err_t esp_wifi_ftm_resp_set_offset (int16_t offset_cm)
Set offset in cm for FTM Responder. An equivalent offset is calculated in picoseconds and added in TOD of FTM Measurement frame (T1).
```

**Attention** Use this API only in AP mode before performing FTM as responder

- offset_cm – T1 Offset to be added in centimeters
- ESP_OK: succeed
- others: failed

```c
esp_err_t esp_wifi_config_11b_rate (wifi_interface_t ifx, bool disable)
Enable or disable 11b rate of specified interface.
```

**Attention** 1. This API should be called after esp_wifi_init() and before esp_wifi_start().
**Attention** 2. Only when really need to disable 11b rate call this API otherwise don’t call this.

- ifx – Interface to be configured.
- disable – true means disable 11b rate while false means enable 11b rate.
- ESP_OK: succeed
- others: failed

```c
esp_err_t esp_wifi_connectionless_module_set_wake_interval (uint16_t wake_interval)
Set wake interval for connectionless modules to wake up periodically.
```

**Attention** 1. Only one wake interval for all connectionless modules.
**Attention** 2. This configuration could work at connected status. When ESP_WIFI_STA_DISCONNECTED_PM_ENABLE is enabled, this configuration could work at disconnected status.
**Attention** 3. Event WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START would be posted each time wake interval starts.
**Attention** 4. Recommend to configure interval in multiples of hundred. (e.g. 100ms)
**Attention** 5. Recommend to configure interval to ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE to get stable performance at coexistence mode.

- wake_interval – Milliseconds after would the chip wake up, from 1 to 65535.

```c
esp_err_t esp_wifi_set_country_code (const char *country, bool ieee80211d_enabled)
configure country
```
Attention 1. When ieee80211d_enabled, the country info of the AP to which the station is connected is used.
E.g. if the configured country is US and the country info of the AP to which the station is connected is JP then the country info that will be used is JP. If the station disconnected from the AP the country info is set back to the country info of the station automatically, US in the example.

Attention 2. When ieee80211d_enabled is disabled, then the configured country info is used always.

Attention 3. When the country info is changed because of configuration or because the station connects to a different external AP, the country IE in probe response/beacon of the soft-AP is also changed.

Attention 4. The country configuration is stored into flash.

Attention 5. When this API is called, the PHY init data will switch to the PHY init data type corresponding to the country info.


Attention 7. When country code “01” (world safe mode) is set, SoftAP mode won’t contain country IE.

Attention 8. The default country is “01” (world safe mode) and ieee80211d_enabled is TRUE.

Attention 9. The third octect of country code string is one of the following: ‘ ‘, ‘O’, ‘I’, ‘X’, otherwise it is considered as ‘ ‘.

参数
- country – the configured country ISO code
- ieee80211d_enabled – 802.11d is enabled or not

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

```
esp_err_t esp_wifi_get_country_code(char *country)
```
get the current country code

参数 country – country code

返回
- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

```
esp_err_t esp_wifi_config_80211_tx_rate(wifi_interface_t ifx, wifi_phy_rate_t rate)
```
Conf 80211 tx rate of specified interface.

Attention 1. This API should be called after esp_wifi_init() and before esp_wifi_start().

参数
- ifx – Interface to be configured.
- rate – Phy rate to be configured.

返回
- ESP_OK: succeed
- others: failed

```
esp_err_t esp_wifi_disable_pmf_config(wifi_interface_t ifx)
```
Disable PMF configuration for specified interface.

Attention This API should be called after esp_wifi_set_config() and before esp_wifi_start().

参数 ifx – Interface to be configured.

返回
• ESP_OK: succeed
• others: failed

Structures

struct wifi_init_config_t
WiFi stack configuration parameters passed to esp_wifi_init call.

Public Members

wifi_osi_funcs_t *osi_funcs
WiFi OS functions

wpa_crypto_funcs_t wpa_crypto_funcs
WiFi station crypto functions when connect

int static_rx_buf_num
WiFi static RX buffer number

int dynamic_rx_buf_num
WiFi dynamic RX buffer number

int tx_buf_type
WiFi TX buffer type

int static_tx_buf_num
WiFi static TX buffer number

int dynamic_tx_buf_num
WiFi dynamic TX buffer number

int cache_tx_buf_num
WiFi TX cache buffer number

int csi_enable
WiFi channel state information enable flag

int ampdu_rx_enable
WiFi AMPDU RX feature enable flag

int ampdu_tx_enable
WiFi AMPDU TX feature enable flag

int amsdu_tx_enable
WiFi AMSDU TX feature enable flag

int nvs_enable
WiFi NVS flash enable flag
int \texttt{nano\_enable}
\hspace{1em}Nano option for printf/scan family enable flag

int \texttt{rx\_ba\_win}
\hspace{1em}WiFi Block Ack RX window size

int \texttt{wifi\_task\_core\_id}
\hspace{1em}WiFi Task Core ID

int \texttt{beacon\_max\_len}
\hspace{1em}WiFi softAP maximum length of the beacon

int \texttt{mgmt\_sbuf\_num}
\hspace{1em}WiFi management short buffer number, the minimum value is 6, the maximum value is 32

\texttt{uint64\_t feature\_caps}
\hspace{1em}Enables additional WiFi features and capabilities

bool \texttt{sta\_disconnected\_pm}
\hspace{1em}WiFi Power Management for station at disconnected status

int \texttt{esp\_now\_max\_encrypt\_num}
\hspace{1em}Maximum encrypt number of peers supported by espnow

int \texttt{magic}
\hspace{1em}WiFi init magic number, it should be the last field

\textbf{Macros}

\texttt{ESP\_ERR\_WIFI\_NOT\_INIT}
\hspace{1em}WiFi driver was not installed by esp_wifi_init

\texttt{ESP\_ERR\_WIFI\_NOT\_STARTED}
\hspace{1em}WiFi driver was not started by esp_wifi_start

\texttt{ESP\_ERR\_WIFI\_NOT\_STOPPED}
\hspace{1em}WiFi driver was not stopped by esp_wifi_stop

\texttt{ESP\_ERR\_WIFI\_IF}
\hspace{1em}WiFi interface error

\texttt{ESP\_ERR\_WIFI\_MODE}
\hspace{1em}WiFi mode error

\texttt{ESP\_ERR\_WIFI\_STATE}
\hspace{1em}WiFi internal state error

\texttt{ESP\_ERR\_WIFI\_CONN}
\hspace{1em}WiFi internal control block of station or soft-AP error
Chapter 2. API 参考

ESP_ERR_WIFI_NVS
WiFi internal NVS module error

ESP_ERR_WIFI_MAC
MAC address is invalid

ESP_ERR_WIFI_SSID
SSID is invalid

ESP_ERR_WIFI_PASSWORD
Password is invalid

ESP_ERR_WIFI_TIMEOUT
Timeout error

ESP_ERR_WIFI_WAKE_FAIL
WiFi is in sleep state(RF closed) and wakeup fail

ESP_ERR_WIFI_WOULD_BLOCK
The caller would block

ESP_ERR_WIFI_NOT_CONNECT
Station still in disconnect status

ESP_ERR_WIFI_POST
Failed to post the event to WiFi task

ESP_ERR_WIFI_INIT_STATE
Invalid WiFi state when init/deinit is called

ESP_ERR_WIFI_STOP_STATE
Returned when WiFi is stopping

ESP_ERR_WIFI_NOT_ASSOC
The WiFi connection is not associated

ESP_ERR_WIFI_TX_DISALLOW
The WiFi TX is disallowed

WIFI_STATIC_TX_BUFFER_NUM

WIFI_CACHE_TX_BUFFER_NUM

WIFI_DYNAMIC_TX_BUFFER_NUM

WIFI_CSI_ENABLED

WIFI_AMPDU_RX_ENABLED
typedef void (*wifi_promiscuous_cb_t)(void *buf, wifi_promiscuous_pkt_type_t type)

The RX callback function in the promiscuous mode. Each time a packet is received, the callback function will be called.

Param buf Data received. Type of data in buffer (wifi_promiscuous_pkt_t or wifi_pkt_rx_ctrl_t) indicated by ‘type’ parameter.

Param type promiscuous packet type.

typedef void (*esp_vendor_ie_cb_t)(void *ctx, wifi_vendor_ie_type_t type, const uint8_t sa[6], const vendor_ie_data_t *vnd_ie, int rssi)

Function signature for received Vendor-Specific Information Element callback.

Param ctx Context argument, as passed to esp_wifi_set_vendor_ie_cb() when registering callback.

Param type Information element type, based on frame type received.

Param sa Source 802.11 address.
Chapter 2. API

### Param vnd_ie
Pointer to the vendor specific element data received.

### Param rssi
Received signal strength indication.

typedef void (*wifi_csi_cb_t)(void *ctx, wifi_csi_info_t *data)

The RX callback function of Channel State Information (CSI) data.

> Each time a CSI data is received, the callback function will be called.

- **Param ctx** context argument, passed to esp_wifi_set_csi_rx_cb() when registering callback function.
- **Param data** CSI data received. The memory that it points to will be deallocated after callback function returns.

#### Header File
- components/esp_wifi/include/esp_wifi_types.h

#### Unions

union wifi_config_t

```c
#include <esp_wifi_types.h>
```

Configuration data for ESP32 AP or STA.

The usage of this union (for ap or sta configuration) is determined by the accompanying interface argument passed to esp_wifi_set_config() or esp_wifi_get_config()

#### Public Members

```c
wifi_ap_config_t ap
```
configuration of AP

```c
wifi_sta_config_t sta
```
configuration of STA

#### Structures

struct wifi_country_t

Structure describing WiFi country-based regional restrictions.

#### Public Members

```c
char cc[3]
```
country code string

```c
uint8_t schan
```
start channel

```c
uint8_t nchan
```
total channel number
### Chapter 2. API

**int8_t max_tx_power**
This field is used for getting WiFi maximum transmitting power, call esp_wifi_set_max_tx_power to set the maximum transmitting power.

**wifi_country_policy_t policy**
country policy

**struct wifi_active_scan_time_t**
Range of active scan times per channel.

#### Public Members

**uint32_t min**
minimum active scan time per channel, units: millisecond

**uint32_t max**
maximum active scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

**struct wifi_scan_time_t**
Aggregate of active & passive scan time per channel.

#### Public Members

**wifi_active_scan_time_t active**
active scan time per channel, units: millisecond.

**uint32_t passive**
passive scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

**struct wifi_scan_config_t**
Parameters for an SSID scan.

#### Public Members

**uint8_t *ssid**
SSID of AP

**uint8_t *bssid**
MAC address of AP

**uint8_t channel**
channel, scan the specific channel

**bool show_hidden**
enable to scan AP whose SSID is hidden
Chapter 2. API

**wifi_scan_type_t**  
scan type, active or passive

**wifi_scan_time_t**  
scan time per channel

**struct wifi_ap_record_t**  
Description of a WiFi AP.

**Public Members**

uint8_t bssid[6]  
MAC address of AP

uint8_t ssid[33]  
SSID of AP

uint8_t primary  
channel of AP

**wifi_second_chan_t**  
secondary channel of AP

int8_t rssi  
signal strength of AP

**wifi_auth_mode_t**  
authmode of AP

**wifi_cipher_type_t**  
pairwise cipher of AP

**wifi_cipher_type_t**  
group_cipher  
group cipher of AP

**wifi_ant_t**  
antenna used to receive beacon from AP

uint32_t phy_11b  
bit: 0 flag to identify if 11b mode is enabled or not

uint32_t phy_11g  
bit: 1 flag to identify if 11g mode is enabled or not

uint32_t phy_11n  
bit: 2 flag to identify if 11n mode is enabled or not
### Chapter 2. API 参考

**uint32_t phy_lr**
- bit: 3 flag to identify if low rate is enabled or not

**uint32_t wps**
- bit: 4 flag to identify if WPS is supported or not

**uint32_t ftmResponder**
- bit: 5 flag to identify if FTM is supported in responder mode

**uint32_t ftm_initiator**
- bit: 6 flag to identify if FTM is supported in initiator mode

**uint32_t reserved**
- bit: 7..31 reserved

**wifi_country_t country**
- country information of AP

**struct wifi_scan_threshold_t**
- Structure describing parameters for a WiFi fast scan.

**Public Members**

**int8_t rssi**
- The minimum rssi to accept in the fast scan mode

**wifi_auth_mode_t authmode**
- The weakest authmode to accept in the fast scan mode. Note: Incase this value is not set and password is set as per WPA2 standards(password len >= 8), it will be defaulted to WPA2 and device won’t connect to deprecated WEP/WPA networks. Please set authmode threshold as WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK to connect to WEP/WPA networks

**struct wifi_pmf_config_t**
- Configuration structure for Protected Management Frame

**Public Members**

**bool capable**
- Deprecated variable. Device will always connect in PMF mode if other device also advertizes PMF capability.

**bool required**
- Advertizes that Protected Management Frame is required. Device will not associate to non-PMF capable devices.

**struct wifi_ap_config_t**
- Soft-AP configuration settings for the ESP32.
Public Members

uint8_t ssid[32]
SSID of ESP32 soft-AP. If ssid_len field is 0, this must be a Null terminated string. Otherwise, length is set according to ssid_len.

uint8_t password[64]
Password of ESP32 soft-AP.

uint8_t ssid_len
Optional length of SSID field.

uint8_t channel
Channel of ESP32 soft-AP

wifi_auth_mode_t authmode
Auth mode of ESP32 soft-AP. Do not support AUTH_WEP in soft-AP mode

uint8_t ssid_hidden
Broadcast SSID or not, default 0, broadcast the SSID

uint8_t max_connection
Max number of stations allowed to connect in, default 4, max 10

uint16_t beacon_interval
Beacon interval which should be multiples of 100. Unit: TU(time unit, 1 TU = 1024 us). Range: 100 ~ 60000. Default value: 100

wifi_cipher_type_t pairwise_cipher
pairwise cipher of SoftAP, group cipher will be derived using this. cipher values are valid starting from WIFI_CIPHER_TYPE_TKIP, enum values before that will be considered as invalid and default cipher suites(TKIP+CCMP) will be used. Valid cipher suites in softAP mode are WIFI_CIPHER_TYPE_TKIP, WIFI_CIPHER_TYPE_CCMP and WIFI_CIPHER_TYPE_TKIP_CCMP.

bool ftm_responder
Enable FTM Responder mode

wifi_pmf_config_t pmf_cfg
Configuration for Protected Management Frame

struct wifi_sta_config_t
STA configuration settings for the ESP32.

Public Members

uint8_t ssid[32]
SSID of target AP.
password[64]
Password of target AP.

scan_method

do all channel scan or fast scan

bool bssid_set

whether set MAC address of target AP or not. Generally, station_config.bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

bssid[6]
MAC address of target AP

channel

channel of target AP. Set to 1~13 to scan starting from the specified channel before connecting to AP. If the channel of AP is unknown, set it to 0.

listen_interval

Listen interval for ESP32 station to receive beacon when WIFI_PS_MAX_MODEM is set. Units: AP beacon intervals. Defaults to 3 if set to 0.

sort_method

sort the connect AP in the list by rssi or security mode

threshold

When sort_method is set, only APs which have an auth mode that is more secure than the selected auth mode and a signal stronger than the minimum RSSI will be used.

pmf_cfg

Configuration for Protected Management Frame. Will be advertised in RSN Capabilities in RSN IE.

rm_enabled

Whether Radio Measurements are enabled for the connection

btm_enabled

Whether BSS Transition Management is enabled for the connection

mbo_enabled

Whether MBO is enabled for the connection

ft_enabled

Whether FT is enabled for the connection

owe_enabled

Whether OWE is enabled for the connection

reserved

Reserved for future feature set
**wifi_sae_pwe_method_t sae_pwe_h2e**

Whether SAE hash to element is enabled

**uint8_t failure_retry_cnt**

Number of connection retries station will do before moving to next AP. scan_method should be set as WIFI_ALL_CHANNEL_SCAN to use this config. Note: Enabling this may cause connection time to increase incase best AP doesn’t behave properly.

**struct wifi_sta_info_t**

Description of STA associated with AP.

**Public Members**

**uint8_t mac[6]**

mac address

**int8_t rssi**

current average rssi of sta connected

**uint32_t phy_11b**

bit: 0 flag to identify if 11b mode is enabled or not

**uint32_t phy_11g**

bit: 1 flag to identify if 11g mode is enabled or not

**uint32_t phy_11n**

bit: 2 flag to identify if 11n mode is enabled or not

**uint32_t phy_lr**

bit: 3 flag to identify if low rate is enabled or not

**uint32_t is_mesh_child**

bit: 4 flag to identify mesh child

**uint32_t reserved**

bit: 5..31 reserved

**struct wifi_sta_list_t**

List of stations associated with the ESP32 Soft-AP.

**Public Members**

**wifi_sta_info_t sta[ESP_WIFI_MAX_CONN_NUM]**

station list

**int num**

number of stations in the list (other entries are invalid)
struct vendor_ie_data_t
Vendor Information Element header.
The first bytes of the Information Element will match this header. Payload follows.

**Public Members**

uint8_t element_id
Should be set to WIFI_VENDOR_IE_ELEMENT_ID (0xDD)

uint8_t length
Length of all bytes in the element data following this field. Minimum 4.

uint8_t vendor_oui[3]
Vendor identifier (OUI).

uint8_t vendor_oui_type
Vendor-specific OUI type.

uint8_t payload[0]
Payload. Length is equal to value in ‘length’ field, minus 4.

struct wifi_pkt_rx_ctrl_t
Received packet radio metadata header, this is the common header at the beginning of all promiscuous mode RX callback buffers.

**Public Members**

signed rssi
Received Signal Strength Indicator (RSSI) of packet. unit: dBm

unsigned rate
PHY rate encoding of the packet. Only valid for non HT(11bg) packet

unsigned __pad0__
reserved

unsigned sig_mode
0: non HT(11bg) packet; 1: HT(11n) packet; 3: VHT(11ac) packet

unsigned __pad1__
reserved

unsigned mcs
Modulation Coding Scheme. If is HT(11n) packet, shows the modulation, range from 0 to 76(MSC0 ~ MCS76)
unsigned `cwb`
Channel Bandwidth of the packet. 0: 20MHz; 1: 40MHz

unsigned `__pad2__`
reserved

unsigned `smoothing`
reserved

unsigned `not_sounding`
reserved

unsigned `__pad3__`
reserved

unsigned `aggregation`
Aggregation. 0: MPDU packet; 1: AMPDU packet

unsigned `stbc`
Space Time Block Code(STBC). 0: non STBC packet; 1: STBC packet

unsigned `fec_coding`
Flag is set for 11n packets which are LDPC

unsigned `sgi`
Short Guide Interval(SGI). 0: Long GI; 1: Short GI

signed `noise_floor`
noise floor of Radio Frequency Module(RF). unit: dBm

unsigned `ampdu_cnt`
ampdu cnt

unsigned `channel`
primary channel on which this packet is received

unsigned `secondary_channel`
secondary channel on which this packet is received. 0: none; 1: above; 2: below

unsigned `__pad4__`
reserved

unsigned `timestamp`
timestamp. The local time when this packet is received. It is precise only if modem sleep or light sleep is not enabled. unit: microsecond

unsigned `__pad5__`
reserved
Chapter 2. API

unsigned __pad6__

reserved

unsigned ant

antenna number from which this packet is received. 0: WiFi antenna 0; 1: WiFi antenna 1

unsigned sig_len

length of packet including Frame Check Sequence(FCS)

unsigned __pad7__

reserved

unsigned rx_state

state of the packet. 0: no error; others: error numbers which are not public

struct wifi_promiscuous_pkt_t

Payload passed to `buf` parameter of promiscuous mode RX callback.

**Public Members**

* wifi_pkt_rx_ctrl_t rx_ctrl

metadata header

* uint8_t payload[0]

Data or management payload. Length of payload is described by rx_ctrl.sig_len. Type of content determined by packet type argument of callback.

struct wifi_promiscuous_filter_t

Mask for filtering different packet types in promiscuous mode.

**Public Members**

* uint32_t filter_mask

OR of one or more filter values WIFI_PROMIS_FILTER_*

struct wifi_csi_config_t

Channel state information(CSI) configuration type.

**Public Members**

* bool lltf_en

enable to receive legacy long training field(lltf) data. Default enabled

* bool htltf_en

enable to receive HT long training field(htltf) data. Default enabled
bool `stbc_htltf2_en`
enable to receive space time block code HT long training field (stbc-htltf2) data. Default enabled

bool `ltf_merge_en`
enable to generate ltf data by averaging lltf and ht_ltf data when receiving HT packet. Otherwise, use ht_ltf data directly. Default enabled

bool `channel_filter_en`
enable to turn on channel filter to smooth adjacent sub-carrier. Disable it to keep independence of adjacent sub-carrier. Default enabled

bool `manu_scale`
manually scale the CSI data by left shifting or automatically scale the CSI data. If set true, please set the shift bits. false: automatically. true: manually. Default false

uint8_t `shift`
manually left shift bits of the scale of the CSI data. The range of the left shift bits is 0~15

struct `wifi_csi_info_t`
CSI data type.

**Public Members**

`wifi_pkt_rx_ctrl_t rx_ctrl`
received packet radio metadata header of the CSI data

uint8_t `mac[6]`
source MAC address of the CSI data

bool `first_word_invalid`
first four bytes of the CSI data is invalid or not

int8_t *`buf`
buffer of CSI data

uint16_t `len`
length of CSI data

struct `wifi_ant_gpio_t`
WiFi GPIO configuration for antenna selection.

**Public Members**

uint8_t `gpio_select`
Whether this GPIO is connected to external antenna switch

uint8_t `gpio_num`
The GPIO number that connects to external antenna switch
struct wifi_ant_gpio_config_t
    WiFi GPIOs configuration for antenna selection.

    Public Members

    wifi_ant_gpio_t gpio_cfg[4]
        The configurations of GPIOs that connect to external antenna switch

struct wifi_ant_config_t
    WiFi antenna configuration.

    Public Members

    wifi_ant_mode_t rx_ant_mode
        WiFi antenna mode for receiving

    wifi_ant_t rx_ant_default
        Default antenna mode for receiving, it’s ignored if rx_ant_mode is not WIFI_ANT_MODE_AUTO

    wifi_ant_mode_t tx_ant_mode
        WiFi antenna mode for transmission, it can be set to WIFI_ANT_MODE_AUTO only if rx_ant_mode is set to WIFI_ANT_MODE_AUTO

    uint8_t enabled_ant0
        Index (in antenna GPIO configuration) of enabled WIFI_ANT_MODE_ANT0

    uint8_t enabled_ant1
        Index (in antenna GPIO configuration) of enabled WIFI_ANT_MODE_ANT1

struct wifi_action_tx_req_t
    Action Frame Tx Request.

    Public Members

    wifi_interface_t ifx
        WiFi interface to send request to

    uint8_t dest_mac[6]
        Destination MAC address

    bool no_ack
        Indicates no ack required

    wifi_action_rx_cb_t rx_cb
        Rx Callback to receive any response
uint32_t data_len
    Length of the appended Data

uint8_t data[0]
    Appended Data payload

struct wifi_ftm_initiator_cfg_t
    FTM Initiator configuration.

Public Members

uint8_t resp_mac[6]
    MAC address of the FTM Responder

uint8_t channel
    Primary channel of the FTM Responder

uint8_t frm_count
    No. of FTM frames requested in terms of 4 or 8 bursts (allowed values - 0(No pref), 16, 24, 32, 64)

uint16_t burst_period
    Requested time period between consecutive FTM bursts in 100’s of milliseconds (0 - No pref)

struct wifi_event_sta_scan_done_t
    Argument structure for WIFI_EVENT_SCAN_DONE event

Public Members

uint32_t status
    status of scanning APs: 0 — success, 1 - failure

uint8_t number
    number of scan results

uint8_t scan_id
    scan sequence number, used for block scan

struct wifi_event_sta_connected_t
    Argument structure for WIFI_EVENT_STA_CONNECTED event

Public Members

uint8_t ssid[32]
    SSID of connected AP
Chapter 2. API

uint8_t **ssid_len**
SSID length of connected AP

uint8_t **bssid[6]**
BSSID of connected AP

uint8_t **channel**
channel of connected AP

**wifi_auth_mode_t authmode**
authentication mode used by AP

```
struct wifi_event_sta_disconnected_t
  Argument structure for WIFI_EVENT_STA_DISCONNECTED event

  **Public Members**

  uint8_t **ssid[32]**
  SSID of disconnected AP

  uint8_t **ssid_len**
  SSID length of disconnected AP

  uint8_t **bssid[6]**
  BSSID of disconnected AP

  uint8_t **reason**
  reason of disconnection

  int8_t **rssi**
  rssi of disconnection

struct wifi_event_sta_authmode_change_t
  Argument structure for WIFI_EVENT_STA_AUTHMODE_CHANGE event

  **Public Members**

  **wifi_auth_mode_t old_mode**
  the old auth mode of AP

  **wifi_auth_mode_t new_mode**
  the new auth mode of AP

struct wifi_event_sta_wps_er_pin_t
  Argument structure for WIFI_EVENT_STA_WPS_ER_PIN event
```
Chapter 2. API Reference

Public Members

uint8_t pin_code[8]
    PIN code of station in enrollee mode

struct wifi_event_sta_wps_er_success_t
    Argument structure for WIFI_EVENT_STA_WPS_ER_SUCCESS event

Public Members

uint8_t ap_cred_cnt
    Number of AP credentials received

uint8_t ssid[MAX_SSID_LEN]
    SSID of AP

uint8_t passphrase[MAX_PASSPHRASE_LEN]
    Passphrase for the AP

struct wifi_event_sta_wps_er_success_t::[anonymous] ap_cred[MAX_WPS_AP_CRED]
    All AP credentials received from WPS handshake

struct wifi_event_ap_staconnected_t
    Argument structure for WIFI_EVENT_AP_STACONNECTED event

Public Members

uint8_t mac[6]
    MAC address of the station connected to ESP32 soft-AP

uint8_t aid
    the aid that ESP32 soft-AP gives to the station connected to

bool is_mesh_child
    flag to identify mesh child

struct wifi_event_ap_stadisconnected_t
    Argument structure for WIFI_EVENT_AP_STADISCONNECTED event

Public Members

uint8_t mac[6]
    MAC address of the station disconnects to ESP32 soft-AP

uint8_t aid
    the aid that ESP32 soft-AP gave to the station disconnects to
bool is_mesh_child
flag to identify mesh child

struct wifi_event_ap_probe_req_rx_t
Argument structure for WIFI_EVENT_AP_PROBEREQRECVED event

Public Members

int rssi
Received probe request signal strength

uint8_t mac[6]
MAC address of the station which send probe request

struct wifi_event_bss_rssi_low_t
Argument structure for WIFI_EVENT_STA_BSS_RSSI_LOW event

Public Members

int32_t rssi
RSSI value of bss

struct wifi_ftm_report_entry_t
Argument structure for

Public Members

uint8_t dlog_token
Dialog Token of the FTM frame

int8_t rssi
RSSI of the FTM frame received

uint32_t rtt
Round Trip Time in pSec with a peer

uint64_t t1
Time of departure of FTM frame from FTM Responder in pSec

uint64_t t2
Time of arrival of FTM frame at FTM Initiator in pSec

uint64_t t3
Time of departure of ACK from FTM Initiator in pSec
Chapter 2. API

uint64_t t4
    Time of arrival of ACK at FTM Responder in pSec

struct wifi_event_ftm_report_t
    Argument structure for WIFI_EVENT_FTM_REPORT event

Public Members

uint8_t peer_mac[6]
    MAC address of the FTM Peer

wifi_ftm_status_t status
    Status of the FTM operation

uint32_t rtt_raw
    Raw average Round-Trip-Time with peer in Nano-Seconds

uint32_t rtt_est
    Estimated Round-Trip-Time with peer in Nano-Seconds

uint32_t dist_est
    Estimated one-way distance in Centi-Meters

wifi_ftm_report_entry_t *ftm_report_data
    Pointer to FTM Report with multiple entries, should be freed after use

uint8_t ftm_report_num_entries
    Number of entries in the FTM Report data

struct wifi_event_action_tx_status_t
    Argument structure for WIFI_EVENT_ACTION_TX_STATUS event

Public Members

wifi_interface_t ifx
    WiFi interface to send request to

uint32_t context
    Context to identify the request

uint8_t da[6]
    Destination MAC address

uint8_t status
    Status of the operation

struct wifi_event_roc_done_t
    Argument structure for WIFI_EVENT_ROC_DONE event
Chapter 2. API 参考

Public Members

```c
uint32_t context
```

Context to identify the request

Macros

```c
WIFI_OFFCHAN_TX_REQ
WIFI_OFFCHAN_TX_CANCEL
WIFI_ROC_REQ
WIFI_ROC_CANCEL
WIFI_PROTOCOL_11B
WIFI_PROTOCOL_11G
WIFI_PROTOCOL_11N
WIFI_PROTOCOL_LR

ESP_WIFI_MAX_CONN_NUM
  max number of stations which can connect to ESP32/ESP32S3/ESP32S2/ESP32C3 soft-AP

WIFI_VENDOR_IE_ELEMENT_ID

WIFI_PROMIS_FILTER_MASK_ALL
  filter all packets

WIFI_PROMIS_FILTER_MASK_MGMT
  filter the packets with type of WIFI_PKT_MGMT

WIFI_PROMIS_FILTER_MASK_CTRL
  filter the packets with type of WIFI_PKT_CTRL

WIFI_PROMIS_FILTER_MASK_DATA
  filter the packets with type of WIFI_PKT_DATA

WIFI_PROMIS_FILTER_MASK_MISC
  filter the packets with type of WIFI_PKT_MISC

WIFI_PROMIS_FILTER_MASK_DATA_MPDU
  filter the MPDU which is a kind of WIFI_PKT_DATA

WIFI_PROMIS_FILTER_MASK_DATA_AMPDU
  filter the AMPDU which is a kind of WIFI_PKT_DATA
```
Chapter 2. API

- `WIFI_PROMIS_FILTER_MASK_FCSFAIL` filter the FCS failed packets, do not open it in general
- `WIFI_PROMIS_CTRL_FILTER_MASK_ALL` filter all control packets
- `WIFI_PROMIS_CTRL_FILTER_MASK_WRAPPER` filter the control packets with subtype of Control Wrapper
- `WIFI_PROMIS_CTRL_FILTER_MASK_BAR` filter the control packets with subtype of Block Ack Request
- `WIFI_PROMIS_CTRL_FILTER_MASK_BA` filter the control packets with subtype of Block Ack
- `WIFI_PROMIS_CTRL_FILTER_MASK_PSPOLL` filter the control packets with subtype of PS-Poll
- `WIFI_PROMIS_CTRL_FILTER_MASK_RTS` filter the control packets with subtype of RTS
- `WIFI_PROMIS_CTRL_FILTER_MASK_CTS` filter the control packets with subtype of CTS
- `WIFI_PROMIS_CTRL_FILTER_MASK_ACK` filter the control packets with subtype of ACK
- `WIFI_PROMIS_CTRL_FILTER_MASK_CFEND` filter the control packets with subtype of CF-END
- `WIFI_PROMIS_CTRL_FILTER_MASK_CFENDACK` filter the control packets with subtype of CF-END+CF-ACK
- `WIFI_EVENT_MASK_ALL` mask all WiFi events
- `WIFI_EVENT_MASK_NONE` mask none of the WiFi events
- `WIFI_EVENT_MASK_AP_PROBEREQRECVED` mask SYSTEM_EVENT_AP_PROBEREQRECVED event

- `MAX_SSID_LEN`
- `MAX_PASSPHRASE_LEN`  
- `MAX_WPS_AP_CRED`
WIFI_STATIS_BUFFER
WIFI_STATIS_RXTX
WIFI_STATIS_HW
WIFI_STATIS_DIAG
WIFI_STATIS_PS
WIFI_STATIS_ALL

Type Definitions

typedef int (*wifi_action_rx_cb_t)(uint8_t*hdr,uint8_t*payload,size_tlen,uint8_t channel)
The Rx callback function of Action Tx operations.

Param hdr pointer to the IEEE 802.11 Header structure
Param payload pointer to the Payload following 802.11 Header
Param len length of the Payload
Param channel channel number the frame is received on

Enumerations

denum wifi_mode_t

Values:

enumerator WIFI_MODE_NULL
null mode

enumerator WIFI_MODE_STA
WiFi station mode

enumerator WIFI_MODE_AP
WiFi soft-AP mode

enumerator WIFI_MODE_APSTA
WiFi station + soft-AP mode

enumerator WIFI_MODE_MAX

enum wifi_interface_t

Values:

enumerator WIFI_IF_STA

enumerator WIFI_IF_AP
enum wifi_country_policy_t
Values:

enumerator WIFI_COUNTRY_POLICY_AUTO
Country policy is auto, use the country info of AP to which the station is connected

enumerator WIFI_COUNTRY_POLICY_MANUAL
Country policy is manual, always use the configured country info

enum wifi_auth_mode_t
Values:

enumerator WIFI_AUTH_OPEN
authenticate mode: open

enumerator WIFI_AUTH_WEP
authenticate mode: WEP

enumerator WIFI_AUTH_WPA_PSK
authenticate mode: WPA_PSK

enumerator WIFI_AUTH_WPA2_PSK
authenticate mode: WPA2_PSK

enumerator WIFI_AUTH_WPA2_WPA2_PSK
authenticate mode: WPA2_WPA2_PSK

enumerator WIFI_AUTH_WPA2_ENTERPRISE
authenticate mode: WPA2_ENTERPRISE

enumerator WIFI_AUTH_WPA3_PSK
authenticate mode: WPA3_PSK

enumerator WIFI_AUTH_WPA2_WPA3_PSK
authenticate mode: WPA2_WPA3_PSK

enumerator WIFI_AUTH_WAPI_PSK
authenticate mode: WAPI_PSK

enumerator WIFI_AUTH_OWE
authenticate mode: OWE

enumerator WIFI_AUTH_MAX

enum wifi_err_reason_t
Values:

enumerator WIFI_REASON_UNSPECIFIED
enumerator WIFI_REASON_AUTH_EXPIRE
enumerator WIFI_REASON_AUTH_LEAVE
enumerator WIFI_REASON_ASSOC_EXPIRE
enumerator WIFI_REASON_ASSOC_TOOMANY
enumerator WIFI_REASON_NOT_AUTHED
enumerator WIFI_REASON_NOT_ASSOCED
enumerator WIFI_REASON_ASSOC_LEAVE
enumerator WIFI_REASON_ASSOC_NOT_AUTHED
enumerator WIFI_REASON_DISASSOC_PWRCAP_BAD
enumerator WIFI_REASON_DISASSOC_SUPCHAN_BAD
enumerator WIFI_REASON_BSS_TRANSITION_DISASSOC
enumerator WIFI_REASON_IE_INVALID
enumerator WIFI_REASON_MIC_FAILURE
enumerator WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT
enumerator WIFI_REASON_GROUP_KEY_UPDATE_TIMEOUT
enumerator WIFI_REASON_IE_IN_4WAY_DIFFERS
enumerator WIFI_REASON_GROUP_CIPHER_INVALID
enumerator WIFI_REASON_PAIRWISE_CIPHER_INVALID
enumerator WIFI_REASON_AKMP_INVALID
enumerator WIFI_REASON_UNSUPP_RSN_IE_VERSION
enumerator WIFI_REASON_INVALID_RSN_IE_CAP
enumerator WIFI_REASON_802_1X_AUTH_FAILED
enumerator WIFI_REASON_CIPHER_SUITE_REJECTED
enumerator WIFI_REASON_TDLS_PEER_UNREACHABLE
enumerator WIFI_REASON_TDLS_UNSPECIFIED
enumerator WIFI_REASON_SSP_REQUESTED_DISASSOC
enumerator WIFI_REASON_NO_SSP_ROAMING_AGREEMENT
enumerator WIFI_REASON_BAD_CIPHER_OR_AKM
enumerator WIFI_REASON_NOT_AUTHORIZED_THIS_LOCATION
enumerator WIFI_REASON_SERVICE_CHANGE_PERCLUDES_TS
enumerator WIFI_REASON_UNSPECIFIED_QOS
enumerator WIFI_REASON_NOT_ENOUGH_BANDWIDTH
enumerator WIFI_REASON_MISSING_ACKS
enumerator WIFI_REASON_EXCEEDED_TXOP
enumerator WIFI_REASON_STA_LEAVING
enumerator WIFI_REASON_END_BA
enumerator WIFI_REASON_UNKNOWN_BA
enumerator WIFI_REASON_TIMEOUT
enumerator WIFI_REASON_PEER_INITIATED
enumerator WIFI_REASON_AP_INITIATED
enumerator WIFI_REASON_INVALID_FT_ACTION_FRAME_COUNT
enumerator WIFI_REASON_INVALID_PMKID
enumerator WIFI_REASON_INVALID_MDE
enumerator WIFI_REASON_INVALID_FTE
enumerator WIFI_REASON_TRANSMISSION_LINK_ESTABLISH_FAILED
enumerator WIFI_REASON_ALTERATIVE_CHANNEL_OCCUPIED
enumerator WIFI_REASON_BEACON_TIMEOUT
enumerator WIFI_REASON_NO_AP_FOUND
enumerator WIFI_REASON_AUTH_FAIL
enumerator WIFI_REASON_ASSOC_FAIL
enumerator WIFI_REASON_HANDSHAKE_TIMEOUT
enumerator WIFI_REASON_CONNECTION_FAIL
enumerator WIFI_REASON_AP_TSF_RESET
enumerator WIFI_REASON_ROAMING

enum wifi_second_chan_t
Values:

enumerator WIFI_SECOND_CHAN_NONE
the channel width is HT20
enumerator WIFI_SECOND_CHAN_ABOVE
the channel width is HT40 and the secondary channel is above the primary channel
enumerator WIFI_SECOND_CHAN_BELOW
the channel width is HT40 and the secondary channel is below the primary channel

enum wifi_scan_type_t
Values:

enumerator WIFI_SCAN_TYPE_ACTIVE
active scan
enumerator WIFI_SCAN_TYPE_PASSIVE
passive scan

enum wifi_cipher_type_t
Values:

enumerator WIFI_CIPHER_TYPE_NONE
the cipher type is none
enumerator WIFI_CIPHER_TYPE_WEP40
the cipher type is WEP40
enumerator WIFI_CIPHER_TYPE_WEP104
the cipher type is WEP104
divider

Chapter 2. API参考

enumerator **WIFI_CIPHER_TYPE_TKIP**
the cipher type is TKIP

enumerator **WIFI_CIPHER_TYPE_CCMP**
the cipher type is CCMP

enumerator **WIFI_CIPHER_TYPE_TKIP_CCMP**
the cipher type is TKIP and CCMP

enumerator **WIFI_CIPHER_TYPE_AES_CMAC128**
the cipher type is AES-CMAC-128

enumerator **WIFI_CIPHER_TYPE_SMS4**
the cipher type is SMS4

enumerator **WIFI_CIPHER_TYPE_GCMP**
the cipher type is GCMP

enumerator **WIFI_CIPHER_TYPE_GCMP256**
the cipher type is GCMP-256

enumerator **WIFI_CIPHER_TYPE_AES_GMAC128**
the cipher type is AES-GMAC-128

enumerator **WIFI_CIPHER_TYPE_AES_GMAC256**
the cipher type is AES-GMAC-256

enumerator **WIFI_CIPHER_TYPE_UNKNOWN**
the cipher type is unknown

divider

enum **wifi_ant_t**
WiFi antenna.

*Values:*
enumerator **WIFI_ANT_ANT0**
WiFi antenna 0
enumerator **WIFI_ANT_ANT1**
WiFi antenna 1
enumerator **WIFI_ANT_MAX**
Invalid WiFi antenna

divider

enum **wifi_scan_method_t**

*Values:*
enumerator **WIFI_FAST_SCAN**
Do fast scan, scan will end after find SSID match AP
enumerator WiFI_ALL_CHANNEL_SCAN
   All channel scan, scan will end after scan all the channel

enum wifi_sort_method_t
   Values:

   enumerator WiFI_CONNECT_AP_BY_SIGNAL
      Sort match AP in scan list by RSSI

   enumerator WiFI_CONNECT_AP_BY_SECURITY
      Sort match AP in scan list by security mode

enum wifi_ps_type_t
   Values:

   enumerator WiFI_PS_NONE
      No power save

   enumerator WiFI_PS_MIN_MODEM
      Minimum modem power saving. In this mode, station wakes up to receive beacon every DTIM period

   enumerator WiFI_PS_MAX_MODEM
      Maximum modem power saving. In this mode, interval to receive beacons is determined by the listen_interval parameter in wifi_sta_config_t

enum wifi_bandwidth_t
   Values:

   enumerator WiFI_BW_HT20

   enumerator WiFI_BW_HT40

enum wifi_sae_pwe_method_t
   Configuration for SAE PWE derivation
   Values:

   enumerator WPA3_SAE_PWE_UNSPECIFIED

   enumerator WPA3_SAE_PWE_HUNT_AND_PECK

   enumerator WPA3_SAE_PWE_HASH_TO_ELEMENT

   enumerator WPA3_SAE_PWE_BOTH

enum wifi_storage_t
   Values:
enumerator **WIFI_STORAGE_FLASH**
all configuration will store in both memory and flash

enumerator **WIFI_STORAGE_RAM**
all configuration will only store in the memory

enum **wifi_vendor_ie_type_t**
Vendor Information Element type.
Determines the frame type that the IE will be associated with.
*Values:*

enumerator **WIFI_VND_IE_TYPE_BEACON**

enumerator **WIFI_VND_IE_TYPE_PROBE_REQ**

enumerator **WIFI_VND_IE_TYPE_PROBE_RESP**

enumerator **WIFI_VND_IE_TYPE_ASSOC_REQ**

enumerator **WIFI_VND_IE_TYPE_ASSOC_RESP**

enum **wifi_vendor_ie_id_t**
Vendor Information Element index.
Each IE type can have up to two associated vendor ID elements.
*Values:*

enumerator **WIFI_VND_IE_ID_0**

enumerator **WIFI_VND_IE_ID_1**

enum **wifi_promiscuous_pkt_type_t**
Promiscuous frame type.
Passed to promiscuous mode RX callback to indicate the type of parameter in the buffer.
*Values:*

enumerator **WIFI_PKT_MGMT**
    Management frame, indicates ‘buf’ argument is wifi_promiscuous_pkt_t

enumerator **WIFI_PKT_CTRL**
    Control frame, indicates ‘buf’ argument is wifi_promiscuous_pkt_t

enumerator **WIFI_PKT_DATA**
    Data frame, indicates ‘buf’ argument is wifi_promiscuous_pkt_t

enumerator **WIFI_PKT_MISC**
    Other type, such as MIMO etc. ‘buf’ argument is wifi_promiscuous_pkt_t but the payload is zero length.
enum wifi_ant_mode_t

WiFi antenna mode.

Values:

enumerator WIFI_ANT_MODE_ANT0
    Enable WiFi antenna 0 only

enumerator WIFI_ANT_MODE_ANT1
    Enable WiFi antenna 1 only

enumerator WIFI_ANT_MODE_AUTO
    Enable WiFi antenna 0 and 1, automatically select an antenna

enumerator WIFI_ANT_MODE_MAX
    Invalid WiFi enabled antenna

enum wifi_phy_rate_t

WiFi PHY rate encodings.

Values:

enumerator WIFI_PHY_RATE_1M_L
    1 Mbps with long preamble

enumerator WIFI_PHY_RATE_2M_L
    2 Mbps with long preamble

enumerator WIFI_PHY_RATE_5M_L
    5.5 Mbps with long preamble

enumerator WIFI_PHY_RATE_11M_L
    11 Mbps with long preamble

enumerator WIFI_PHY_RATE_2M_S
    2 Mbps with short preamble

enumerator WIFI_PHY_RATE_5M_S
    5.5 Mbps with short preamble

enumerator WIFI_PHY_RATE_11M_S
    11 Mbps with short preamble

enumerator WIFI_PHY_RATE_48M
    48 Mbps

enumerator WIFI_PHY_RATE_24M
    24 Mbps
enumerator WIFI_PHY_RATE_12M
   12 Mbps
enumerator WIFI_PHY_RATE_6M
   6 Mbps
enumerator WIFI_PHY_RATE_54M
   54 Mbps
enumerator WIFI_PHY_RATE_36M
   36 Mbps
enumerator WIFI_PHY_RATE_18M
   18 Mbps
enumerator WIFI_PHY_RATE_9M
   9 Mbps
enumerator WIFI_PHY_RATE_MCS0_LGI
   MCS0 with long GI, 6.5 Mbps for 20MHz, 13.5 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS1_LGI
   MCS1 with long GI, 13 Mbps for 20MHz, 27 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS2_LGI
   MCS2 with long GI, 19.5 Mbps for 20MHz, 40.5 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS3_LGI
   MCS3 with long GI, 26 Mbps for 20MHz, 54 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS4_LGI
   MCS4 with long GI, 39 Mbps for 20MHz, 81 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS5_LGI
   MCS5 with long GI, 52 Mbps for 20MHz, 108 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS6_LGI
   MCS6 with long GI, 58.5 Mbps for 20MHz, 121.5 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS7_LGI
   MCS7 with long GI, 65 Mbps for 20MHz, 135 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS0_SGI
   MCS0 with short GI, 7.2 Mbps for 20MHz, 15 Mbps for 40MHz
enumerator WIFI_PHY_RATE_MCS1_SGI
   MCS1 with short GI, 14.4 Mbps for 20MHz, 30 Mbps for 40MHz
Chapter 2. API

enumerator `WIFI_PHY_RATE_MCS2_SGI`
MCS2 with short GI, 21.7 Mbps for 20MHz, 45 Mbps for 40MHz

enumerator `WIFI_PHY_RATE_MCS3_SGI`
MCS3 with short GI, 28.9 Mbps for 20MHz, 60 Mbps for 40MHz

enumerator `WIFI_PHY_RATE_MCS4_SGI`
MCS4 with short GI, 43.3 Mbps for 20MHz, 90 Mbps for 40MHz

enumerator `WIFI_PHY_RATE_MCS5_SGI`
MCS5 with short GI, 57.8 Mbps for 20MHz, 120 Mbps for 40MHz

enumerator `WIFI_PHY_RATE_MCS6_SGI`
MCS6 with short GI, 65 Mbps for 20MHz, 135 Mbps for 40MHz

enumerator `WIFI_PHY_RATE_MCS7_SGI`
MCS7 with short GI, 72.2 Mbps for 20MHz, 150 Mbps for 40MHz

enumerator `WIFI_PHY_RATE_LORA_250K`
250 Kbps

enumerator `WIFI_PHY_RATE_LORA_500K`
500 Kbps

enumerator `WIFI_PHY_RATE_MAX`

enum `wifi_event_t`

WiFi event declarations

Values:

enumerator `WIFI_EVENT_WIFI_READY`
ESP32 WiFi ready

enumerator `WIFI_EVENT_SCAN_DONE`
ESP32 finish scanning AP

enumerator `WIFI_EVENT_STA_START`
ESP32 station start

enumerator `WIFI_EVENT_STA_STOP`
ESP32 station stop

enumerator `WIFI_EVENT_STA_CONNECTED`
ESP32 station connected to AP

enumerator `WIFI_EVENT_STA_DISCONNECTED`
ESP32 station disconnected from AP
enumerator **WIFI_EVENT_STA_AUTHMODE_CHANGE**
the auth mode of AP connected by ESP32 station changed

enumerator **WIFI_EVENT_STA_WPS_ER_SUCCESS**
ESP32 station wps succeeds in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_FAILED**
ESP32 station wps fails in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_TIMEOUT**
ESP32 station wps timeout in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_PIN**
ESP32 station wps pin code in enrollee mode

enumerator **WIFI_EVENT_STA_WPS_ER_PBC_OVERLAP**
ESP32 station wps overlap in enrollee mode

enumerator **WIFI_EVENT_AP_START**
ESP32 soft-AP start

enumerator **WIFI_EVENT_AP_STOP**
ESP32 soft-AP stop

enumerator **WIFI_EVENT_AP_STACONNECTED**
a station connected to ESP32 soft-AP

enumerator **WIFI_EVENT_AP_STADISCONNECTED**
a station disconnected from ESP32 soft-AP

enumerator **WIFI_EVENT_AP_PROBEREQRECEIVED**
Receive probe request packet in soft-AP interface

enumerator **WIFI_EVENT_FTM_REPORT**
Receive report of FTM procedure

enumerator **WIFI_EVENT_STA_BSS_RSSI_LOW**
AP’s RSSI crossed configured threshold

enumerator **WIFI_EVENT_ACTION_TX_STATUS**
Status indication of Action Tx operation

enumerator **WIFI_EVENT_ROC_DONE**
Remain-on-Channel operation complete

enumerator **WIFI_EVENT_STA_BEACON_TIMEOUT**
ESP32 station beacon timeout
Chapter 2. API

enumerator WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START
    ESP32 connectionless module wake interval start

enumerator WIFI_EVENT_MAX
    Invalid WiFi event ID

enum wifi_event_sta_wps_fail_reason_t
    Argument structure for WIFI_EVENT_STA_WPS_ER_FAILED event
    Values:

enumerator WPS_FAIL_REASON_NORMAL
    ESP32 WPS normal fail reason

enumerator WPS_FAIL_REASON_RECV_M2D
    ESP32 WPS receive M2D frame

enumerator WPS_FAIL_REASON_MAX

enum wifi_ftm_status_t
    FTM operation status types.
    Values:

enumerator FTM_STATUS_SUCCESS
    FTM exchange is successful

enumerator FTM_STATUS_UNSUPPORTED
    Peer does not support FTM

enumerator FTM_STATUS_CONF_REJECTED
    Peer rejected FTM configuration in FTM Request

enumerator FTM_STATUS_NO_RESPONSE
    Peer did not respond to FTM Requests

enumerator FTM_STATUS_FAIL
    Unknown error during FTM exchange

Wi-Fi Easy Connect™ (DPP)

Wi-Fi Easy Connect™, also known as Device Provisioning Protocol (DPP) or Easy Connect, is a provisioning protocol certified by Wi-Fi Alliance. It is a secure and standardized provisioning protocol for configuration of Wi-Fi Devices. With Easy Connect adding a new device to a network is as simple as scanning a QR Code. This reduces complexity and enhances user experience while onboarding devices without UI like Smart Home and IoT products. Unlike old protocols like WiFi Protected Setup (WPS), Wi-Fi Easy Connect incorporates strong encryption through public key cryptography to ensure networks remain secure as new devices are added. Easy Connect brings many benefits in the User Experience:

- Simple and intuitive to use; no lengthy instructions to follow for new device setup
- No need to remember and enter passwords into the device being provisioned
- Works with electronic or printed QR codes, or human-readable strings
• Supports both WPA2 and WPA3 networks

Please refer to Wi-Fi Alliance’s official page on Easy Connect for more information.

ESP32 supports Enrollee mode of Easy Connect with QR Code as the provisioning method. A display is required to display this QR Code. Users can scan this QR Code using their capable device and provision the ESP32 to their Wi-Fi network. The provisioning device needs to be connected to the AP which need not support Wi-Fi Easy Connect™. Easy Connect is still an evolving protocol. Of known platforms that support the QR Code method are some Android smartphones with Android 10 or higher. To use Easy Connect no additional App needs to be installed on the supported smartphone.

**Application Example** Example on how to provision ESP32 using a supported smartphone:

```bash
code
wifi/wifi_easy_connect/dpp-enrollee
```

**API Reference**

**Header File**

- `components/wpa_supplicant/esp_supplicant/include/esp_dpp.h`

**Functions**

```c
esp_err_t esp_supp_dpp_init (esp_supp_dpp_event_cb_t evt_cb)
```

Initializes DPP Supplicant.

```
Starts DPP Supplicant and initializes related Data Structures.
```

```
return
```

- `ESP_OK`: Success
- `ESP_FAIL`: Failure

```c
esp_err_t esp_supp_dpp_bootstrap_gen (const char* chan_list, esp_supp_dpp_bootstrap_t type, const char* key, const char* info)
```

Generates Bootstrap Information as an Enrollee.

```
Generates Out Of Band Bootstrap information as an Enrollee which can be used by a DPP Configurator to provision the Enrollee.
```

```
参数
```

- `chan_list` - List of channels device will be available on for listening
- `type` - Bootstrap method type, only QR Code method is supported for now.
- `key` - (Optional) 32 byte Raw Private Key for generating a Bootstrapping Public Key
- `info` - (Optional) Ancilliary Device Information like Serial Number

```
返回
```
Chapter 2. API 参考

- ESP_OK: Success
- ESP_FAIL: Failure

`esp_err_t esp_supp_dpp_start_listen(void)`
Start listening on Channels provided during esp_supp_dpp_bootstrap_gen.

```
Listens on every Channel from Channel List for a pre-defined wait time.
```

- ESP_OK: Success
- ESP_FAIL: Generic Failure
- ESP_ERR_INVALID_STATE: ROC attempted before WiFi is started
- ESP_ERR_NO_MEM: Memory allocation failed while posting ROC request

`void esp_supp_dpp_stop_listen(void)`
Stop listening on Channels.

```
Stops listening on Channels and cancels ongoing listen operation.
```

**Macros**

**ESP_ERR_DPP_FAILURE**
Generic failure during DPP Operation

**ESP_ERR_DPP_TX_FAILURE**
DPP Frame Tx failed OR not Acked

**ESP_ERR_DPP_INVALID_ATTR**
Encountered invalid DPP Attribute

**Type Definitions**

typedef enum `dpp_bootstrap_type` `esp_supp_dpp_bootstrap_t`
Types of Bootstrap Methods for DPP.

typedef void (`*esp_supp_dpp_event_cb_t`)(`esp_supp_dpp_event_t` evt, void *data)
Callback function for receiving DPP Events from Supplicant.

```
Callback function will be called with DPP related information.
```

- Param `evt` DPP event ID
- Param `data` Event data payload

**Enumerations**

type `enum dpp_bootstrap_type`
Types of Bootstrap Methods for DPP.

- Values:
enumerator DPP_BOOTSTRAP_QR_CODE
QR Code Method

enumerator DPP_BOOTSTRAP_PKEX
Proof of Knowledge Method

enumerator DPP_BOOTSTRAP_NFC_URI
NFC URI record Method

enum esp_supp_dpp_event_t
Types of Callback Events received from DPP Supplicant.
Values:

enumerator ESP_SUPP_DPP_URI_READY
URI is ready through Bootstrapping

enumerator ESP_SUPP_DPP_CFG_RECVD
Config received via DPP Authentication

enumerator ESP_SUPP_DPP_FAIL
DPP Authentication failure

本部分的 Wi-Fi API 示例代码存放在 ESP-IDF 示例项目的 wifi 目录下。ESP-WIFI-MESH 的示例代码存放在 ESP-IDF 示例项目的 mesh 目录下。

2.5.2 以太网

以太网

概述 ESP-IDF 提供一系列功能强大且兼具一致性的 API，为内部以太网 MAC (EMAC) 控制器和外部 SPI-Ethernet 模块提供支持。

本编程指南分为以下几个部分：
1. 以太网基本概念
2. 配置 MAC 和 PHY
3. 连接驱动程序至 TCP/IP 协议栈
4. 以太网驱动程序的条目控制

以太网基本概念 以太网是一种异步的带冲突检测的载波侦听多路访问 (CSMA/CD) 协议接口。通常来说，以太网不太适用于低功率应用。然而，得益于其广泛的部署、高效的网络连接、高数据率以及范围不限的可扩展性，几乎所有的有线通信都可以通过以太网进行。

符合 IEEE 802.3 标准的正常以太网帧的长度在 64 至 1518 字节之间，由五个或六个不同的字段组成：目的地 MAC 地址 (DA)、源 MAC 地址 (SA)、类型/长度字段、数据有效载荷字段、可选的填充字段和帧校验序列字段 (CRC)。此外，在以太网上传输时，以太网数据包的开头需附加 7 字节的前导码和 1 字节的帧起始符 (SOF)。

因此，双绞线上的通信如图所示:
### Ethernet Data Frame Format

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Preamble (7 Bytes)</td>
</tr>
<tr>
<td>2</td>
<td>Start-of-Frame Delimiter (1 Byte)</td>
</tr>
<tr>
<td>3</td>
<td>Destination Address (6 Bytes)</td>
</tr>
<tr>
<td>4</td>
<td>Source Address (6 Bytes)</td>
</tr>
<tr>
<td>5</td>
<td>Type / Length (2 Bytes)</td>
</tr>
<tr>
<td>6</td>
<td>Payload (0 ~ 1500 Bytes)</td>
</tr>
<tr>
<td>7</td>
<td>Pad (if necessary)</td>
</tr>
<tr>
<td>8</td>
<td>Frame Check Sequence (4 Bytes)</td>
</tr>
</tbody>
</table>

图 4: 以太网数据帧格式

**前导码和帧起始符** 前导码包含 7 字节的 55H，作用是使接收器在实际帧到达之前锁定数据流。

帧前界定符 (SFD) 为二进制序列 10101011（物理介质层可见），有时它也被视作前导码的一部分。

在传输和接收数据时，协议将自动从数据包中生成/移除前导码和帧起始符。

**目的地址 (DA)** 目的地址字段包含一个 6 字节长的设备 MAC 地址，数据包将发送到该地址。如果 MAC 地址第一个字节中的最低有效位是 1，则该地址为组播地址。例如，01-00-00-F0-00 和 33-45-67-89-AB-CD 是组播地址，而 00-00-00-F0-00 和 32-45-67-89-AB-CD 不是。

带有组播地址的数据包将到达选定的一组以太网节点，并发挥重要作用。如果目的地址字段是保留的多播地址，即 FF-FF-FF-FF-FF-FF，则该数据包是一个广播数据包，指向共享网络中的每个对象。如果 MAC 地址的第一个字节中的最低有效位为 0，则该地址为单播地址，仅供寻址节点使用。

通常，EMAC 控制器会集成接收过滤器，用于丢弃或接收带有组播、广播或单播目的地址的数据包。传输数据包时，由主机控制器将所需的目标地址写入传输缓冲区。

**源地址 (SA)** 源地址字段包含一个 6 字节长的节点 MAC 地址，以太网数据包通过该节点创建。以太网的用户需为所使用的任意控制生成唯一的 MAC 地址。MAC 地址由两部分组成：前三个字节称为组织唯一标识符 (OUI)，由 IEEE 分配；后三个字节是地址字节，由购买 OUI 的公司配置。有关 ESP-IDF 中使用的 MAC 地址的详细信息，请参阅 MAC 地址分配。

传输数据包时，由主机控制器将分配的源 MAC 地址写入传输缓冲区。

**类型/长度** 类型/长度字段长度为 2 字节。如果其值 <= 1500 （十进制），则该字段为长度字段，指定在数据字段后的非填充数据量；如果其值 >= 1536，则该字段值表示后续数据包所属的协议。以下为该字段的常见值：

- IPv4 = 0800H
Chapter 2. API 参考

- IPv6 = 86DDH
- ARP = 0806H

使用专有网络的用户可以将此字段配置为长度字段。然而，对于使用互联网协议 (IP) 或地址解析协议 (ARP) 等协议的应用程序，在传输数据包时，应将此字段配置为协议规范定义的适当类型。

数据有效载荷 数据有效载荷字段是一个可变长度的字段，长度从 0 到 1500 字节不等。更大的数据包会因违反以太网标准而被大多数以太网节点丢弃。

数据有效载荷字段包含客户端数据，如 IP 数据报。

填充及帧校验序列 (FCS) 填充字段是一个可变长度的字段。数据有效载荷较小时，将添加填充字段以满足 IEEE 802.3 规范的要求。

以太网数据包的 DA、SA、类型、数据有效载荷和填充字段共计必须不少于 60 字节。加上所需的 4 字节 FCS 字段，数据包的长度必须不少于 64 字节。如果数据有效载荷字段小于 46 字节，则需要加上一个填充字段。

帧校验序列字段 (FCS) 长度为 4 字节，其中包含一个行业标准的 32 位 CRC，该 CRC 是根据 DA、SA、类型、数据有效载荷和填充字段的数据计算的。鉴于计算 CRC 的复杂性，硬件通常会自动生成一个有效的 CRC 进行传输。否则，需由主机控制器生成 CRC 并将其写入传输缓冲区。

通常情况下，主机控制器无需关注填充字段和 CRC 字段，因为这两部分可以在传输或接收时由硬件 EMAC 自动生成或验证。然而，当数据包到达时，填充字段和 CRC 字段将被写入接收缓冲区。因此，如果需要的话，主机控制器也可以对它们进行评估。

备注：除了上述的基本数据帧，在 10/100 Mbps 以太网中还有两种常见的帧类型：控制帧和 VLAN 标记帧。ESP-IDF 不支持这两种帧类型。

配置 MAC 和 PHY 以太网驱动器由两部分组成：MAC 和 PHY。

MAC 和 PHY 之间的通信可以通过多种方式进行，如：MII（媒体独立接口）、RMII（简化媒体独立接口）等。

MII 和 RMII 的一个明显区别在于其所需的信号数。MII 通常需要多达 18 个信号，RMII 接口则仅需要 9 个信号。

在 RMII 模式下，接收器和发送器信号的参考时钟为 REF_CLK。在访问 PHY 和 MAC 时，REF_CLK 必须保持稳定，一般来说，根据您设计的 PHY 设备的特征，可通过对以下三种方式生成 REF_CLK：

- 一些 PHY 芯片可以从此外部连接的 25 MHz 晶体振荡器中获取 REF_CLK（如图中的选项 a 所示）。对于此类芯片，请在 CONFIG_ETH_RMII_CLK_MODE 中选择 CONFIG_ETH_RMII_CLK_INPUT。
- 一些 PHY 芯片使用可以作为 MAC 端 REF_CLK 的外接 50 MHz 晶体振荡器或其他时钟源（如图中的选项 b 所示）。对于此类芯片，请同样在 CONFIG_ETH_RMII_CLK_MODE 中选择 CONFIG_ETH_RMII_CLK_INPUT。
- 一些 EMAC 控制器可以使用其内部的高精度 PLL 生成 REF_CLK（如图中的选项 c 所示）。此种情况下，请在 CONFIG_ETH_RMII_CLK_MODE 中选择 CONFIG_ETH_RMII_CLK_OUTPUT。

备注：如上所述，REF_CLK 默认通过项目配置进行配置。然而，通过设置 eth.esp32_emac_config_t::interface 和 eth.esp32_emac_config_t::clock_config，也可以实现在用户应用代码中覆盖该时钟。更多细节，请参见 emac_rmii_clock_mode_t 和 emac_rmii_clock_gpio_t。

警告：如果配置 RMII 时钟模式为 CONFIG_ETH_RMII_CLK_OUTPUT，那么就可以使用 GPIO0 输出 REF_CLK 信号。更多细节，请参见 CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0。
Chapter 2. API 参考

图 5: 以太网 RMII 接口
Chapter 2. API 参考

值得一体的是，如果您在设计中并未使用 PSRAM，则 GPIO16 和 GPIO17 也可以用来输出参考时钟。更多细节，请参见 CONFIG_ETH_RMI_CLK_OUT_GPIO。

如果配置 RMII 时钟模式为 CONFIG_ETH_RMII_CLK_INPUT，那么有且只有 GPIO0 可以用来输入 REF_CLK 信号。请注意，GPIO0 同时也是 ESP32 上一个重要的 strapping GPIO 管脚。如果 GPIO0 在上电时来样为低电平，ESP32 将进入下载模式，需进行手动复位重启系统。解决这个问题的方法是，在硬件中默认禁用 REF_CLK，从而避免 strapping 管脚在启动阶段受到其他信号的干扰。随后，再在以太网驱动安装阶段重新启用 REF_CLK。

可以通过以下方法禁用 REF_CLK 信号：

- 禁用或关闭晶体谐振器的电（对数图中的项 b）。
- 强制复位 PHY 设备（对图中的项 a）。此方法并不适用于所有 PHY 设备，即便处于复位状态，某些 PHY 设备仍会向 GPIO0 输出信号。

无论选择哪种 RMII 时钟模式，都请确保硬件设计中 REF_CLK 的信号完整性！信号线越短越好，并请保持信号线与 RF 设备和天线距离的距离。

备注：ESP-IDF 只支持 RMII 接口（即在 Kconfig 选项 CONFIG_ETH_PHY_INTERFACE 中始终选择 CONFIG_ETH_PHY_INTERFACE_RMII）。

在数据平面使用的信号通过 MUX 连接至特定的 GPIO。这些信号无法配置至其他 GPIO。在控制平面使用的信号则可以通过 Matrix 矩阵路由到任何空闲 GPIO。相关的硬件设计示例，请参考 ESP32-Ethernet-Kit。

根据您的以太网板设计，需要分别为 MAC 和 PHY 配置必要的参数。通过两个完成驱动程序的安装。

MAC 相关配置可以在 eth_mac_config_t 中找到，具体包括：

- `eth_mac_config_t::sw_reset_timeout_ms`：软件复位超时值，单位为毫秒。通常，MAC 复位应为 100ms 内完成。
- `eth_mac_config_t::rx_task_stack_size` 和 `eth_mac_config_t::rx_task_prio`。MAC 驱动会创建一个任务来处理传入的数据包，这两个参数用于设置该任务的堆栈大小和优先级。
- `eth_mac_config_t::flags`：指定 MAC 驱动应支持的额外功能，尤其适用于某些特殊情况。这个字段的值支持以 ETH_MAC_FLAG_ 为前缀的宏进行 OR 运算。例如，如果 MAC 驱动应禁用缓存后开始工作，那么则需要用 `ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE` 配置这个字段。
- `eth_esp32_emac_config_t::smi_mdio_gpio_num` 和 `eth_esp32_emac_config_t::smi_mdio_gpio_num`。连接 SMI 信号的 GPIO 编号。
- `eth_esp32_emac_config_t::interface`：配置到 PHY（MII/RMII）的 MAC 数据接口。
- `eth_esp32_emac_config_t::clock_config`：配置 EMAC 接口时钟（RMII 模式下的 REF_CLK 模式以及 GPIO0 引脚）。

MAC 相关配置可以在 eth_phy_config_t 中找到，具体包括：

- `eth_phy_config_t::phy_addr`：同一条 SMI 总线上可以存在多个 PHY 设备。所以有必要为各个 PHY 设备分配唯一地址。通常，这个地址是在硬件设计期间，通过拉高/拉低一些 PHY strapping 管脚来分配的。根据不同的以太网开发板，可配置值为 0 到 15。需注意，如果 SMI 总线上仅有一个 PHY 设备，将该值配置为 -1，即可使驱动程序自动检测 PHY 地址。
- `eth_phy_config_t::reset_timeout_ms`：复位超时值，单位为毫秒。通常，PHY 复位应在 100ms 内完成。
- `eth_phy_config_t::autonego_timeout_ms`：自动协商超时值，单位为毫秒。以太网驱动程序会自动与对等的以太网节点进行协商，以确定双工和速模式。此值通常取决于您电路板上 PHY 设备的性能。
- `eth_phy_config_t::reset_gpio_num`：如果您的开发板同时将 PHY 复位管脚连接至了任意 GPIO 管脚，请使用该字段进行配置。否则，配置为 -1。
ESP-IDF 在宏 `ETH_MAC_DEFAULT_CONFIG` 和 `ETH_PHY_DEFAULT_CONFIG` 中为 MAC 和 PHY 提供了默认配置。

## 创建 MAC 和 PHY 实例
以太网驱动是以面向对象的方式实现的。对 MAC 和 PHY 的任何操作都应基于实例。

### 内部 MAC + 外部 PHY

```c
eth_mac_config_t mac_config = ETH_MAC_DEFAULT_CONFIG();      // 应用默认的 MAC 配置
mac_config.smii_mdc_gpio_num = CONFIG_EXAMPLE_ETH_MDC_GPIO;  // 更改用于 MDC
                // 信号的 GPIO
mac_config.smii_mdio_gpio_num = CONFIG_EXAMPLE_ETH_MDI0_GPIO;  // 更改用于 MDIO
                // 信号的 GPIO
esp_eth_mac_t *mac = esp_eth_mac_new_esp32(&mac_config);     // 创建 MAC 实例

eth_phy_config_t phy_config = ETH_PHY_DEFAULT_CONFIG();        // 应用默认的 PHY 配置
phy_config.phy_addr = CONFIG_EXAMPLE_ETH_PHY_ADDR;            // 根据开发板设计更改
                // PHY 地址
phy_config.reset_gpio_num = CONFIG_EXAMPLE_ETH_PHY_RST_GPIO;  // 更改用于 PHY
                // 复位的 GPIO
esp_eth_phy_t *phy = esp_eth_phy_new_ip101(&phy_config);      // 创建 PHY 实例
                // ESP-IDF 为数种以太网 PHY 芯片驱动提供官方支持
                // esp_eth_phy_t *phy = esp_eth_phy_new_rt18201(&phy_config);
                // esp_eth_phy_t *phy = esp_eth_phy_new_lan8720(&phy_config);
                // esp_eth_phy_t *phy = esp_eth_phy_new_dp83848(&phy_config);
```

### 可选的运行时 MAC 时钟配置
可以通过用户应用程序代码，选择性配置 EMAC 中的 REF_CLK。

```c
eth_mac_config_t mac_config = ETH_MAC_DEFAULT_CONFIG();      // 应用默认的 MAC 配置
                // ...
mac_config.interface = EMAC_DATA_INTERFACE_RMII;             // 更改 EMAC 数据接口
mac_config.clock_config.rmii.clock_mode = EMAC_CLK_OUT;     // 配置 EMAC REF_CLK 模式
mac_config.clock_config.rmii.clock_gpio = EMAC_CLK_OUT_GPIO; // 配置用于输入/输出
                // EMAC REF_CLK 的 GPIO 编号
esp_eth_mac_t *mac = esp_eth_mac_new_esp32(&mac_config);    // 创建 MAC 实例
```

### SPI-Ethernet 模块

```c
eth_mac_config_t mac_config = ETH_MAC_DEFAULT_CONFIG();      // 应用默认的 MAC 配置
eth_phy_config_t phy_config = ETH_PHY_DEFAULT_CONFIG();      // 应用默认的 PHY 配置
phy_config.phy_addr = CONFIG_EXAMPLE_ETH_PHY_ADDR;           // 根据开发板设计更改
                // PHY 地址
phy_config.reset_gpio_num = CONFIG_EXAMPLE_ETH_PHY_RST_GPIO; // 更改用于 PHY
                // 复位的 GPIO
                // 安装 GPIO 中断服务（因为 SPI-Ethernet 模块为中断驱动）
gpio_install_isr_service(0);
                // 配置 SPI 总线
spi_device_handle_t spi_handle = NULL;
spi_bus_config_t buscfg = {
    .miso_io_num = CONFIG_EXAMPLE_ETH_SPI_MISO_GPIO,
    .mosi_io_num = CONFIG_EXAMPLE_ETH_SPI_MOSI_GPIO,
    .sclk_io_num = CONFIG_EXAMPLE_ETH_SPI_SCLK_GPIO,
    .quadwp_io_num = -1,
    .quadhd_io_num = -1,
};
ESP_ERROR_CHECK(spi_bus_initialize(CONFIG_EXAMPLE_ETH_SPI_HOST, &buscfg, 1));
                // 配置 SPI 从机设备
```

(下页继续)
Chapter 2. API 参考

(续上页)

spi_device_interface_config_t spi_devcfg = {
.mode = 0,
.clock_speed_hz = CONFIG_EXAMPLE_ETH_SPI_CLOCK_MHZ * 1000 * 1000,
.spics_io_num = CONFIG_EXAMPLE_ETH_SPI_CS_GPIO,
.queue_size = 20
};
/* dm9051 ethernet driver is based on spi driver */
eth_dm9051_config_t dm9051_config = ETH_DM9051_DEFAULT_CONFIG(CONFIG_EXAMPLE_ETH_
,→SPI_HOST, &spi_devcfg);
dm9051_config.int_gpio_num = CONFIG_EXAMPLE_ETH_SPI_INT_GPIO;
esp_eth_mac_t *mac = esp_eth_mac_new_dm9051(&dm9051_config, &mac_config);
esp_eth_phy_t *phy = esp_eth_phy_new_dm9051(&phy_config);

备注:
• 当为 SPI-Ethernet 模块（例如 DM9051）创建 MAC 和 PHY 实例时，由于 PHY 是集成在模块中的，因此
调用的实例创建函数的后缀须保持一致（例如 esp_eth_mac_new_dm9051 和 esp_eth_phy_new_dm9051
搭配使用）
。
• 针 对 不 同 的 以 太 网 模 块， 或 是 为 了 满 足 特 定 PCB 上 的 SPI 时 序，SPI 从 机 设 备 配 置 （即
spi_device_interface_config_t）可能略有不同。具体配置请查看模块规格以及 ESP-IDF 中的示例。

安装驱动程序 安装以太网驱动程序需要结合 MAC 和 PHY 实例，并在esp_eth_config_t 中配置一
些额外的高级选项（即不仅限于 MAC 或 PHY 的选项）：

• esp_eth_config_t::mac：由 MAC 生成器创建的实例（例如esp_eth_mac_new_esp32()）
。
• esp_eth_config_t::phy：由 PHY 生成器创建的实例（例如esp_eth_phy_new_ip101()）。
• esp_eth_config_t::check_link_period_ms：以太网驱动程序会启用操作系统定时器来定
期检查链接状态。该字段用于设置间隔时间，单位为毫秒。
• esp_eth_config_t::stack_input：在大多数的以太网物联网应用中，驱动器接收的以太网
帧会被传递到上层（如 TCP/IP 栈）
。经配置，该字段为负责处理传入帧的函数。您可以在安装驱动
程序后，通过函数esp_eth_update_input_path() 更新该字段。该字段支持在运行过程中进
行更新。
• esp_eth_config_t::on_lowlevel_init_done 和esp_eth_config_t::on_lowlevel_deinit_done：
这两个字段用于指定钩子函数，当去初始化或初始化低级别硬件时，会调用钩子函数。
ESP-IDF 在宏ETH_DEFAULT_CONFIG 中为安装驱动程序提供了一个默认配置。
esp_eth_config_t config = ETH_DEFAULT_CONFIG(mac, phy); // 应 用 默 认 驱 动 程 序 配 置
esp_eth_handle_t eth_handle = NULL; // 驱 动 程 序 安 装 完 毕 后 ， 将 得 到 驱 动 程 序 的 句 柄
esp_eth_driver_install(&config, &eth_handle); // 安 装 驱 动 程 序

以太网驱动程序包含事件驱动模型，该模型会向用户空间发送有用及重要的事件。安装以太网驱动程序
之前，需要首先初始化事件循环。有关事件驱动编程的更多信息，请参考ESP Event.
/** 以 太 网 事 件 的 事 件 处 理 程 序 */
static void eth_event_handler(void *arg, esp_event_base_t event_base,
int32_t event_id, void *event_data)
{
uint8_t mac_addr[6] = {0};
/* 可 从 事 件 数 据 中 获 得 以 太 网 驱 动 句 柄 */
esp_eth_handle_t eth_handle = *(esp_eth_handle_t *)event_data;
switch (event_id) {
case ETHERNET_EVENT_CONNECTED:
esp_eth_ioctl(eth_handle, ETH_CMD_G_MAC_ADDR, mac_addr);
ESP_LOGI(TAG, "Ethernet Link Up");
ESP_LOGI(TAG, "Ethernet HW Addr %02x:%02x:%02x:%02x:%02x:%02x",
mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_
,→addr[4], mac_addr[5]);
(下页继续)

Espressif Systems

894
Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151


break;
case ETHERNET_EVENT_DISCONNECTED:
    ESP_LOGI(TAG, "Ethernet Link Down");
    break;
case ETHERNET_EVENT_START:
    ESP_LOGI(TAG, "Ethernet Started");
    break;
case ETHERNET_EVENT_STOP:
    ESP_LOGI(TAG, "Ethernet Stopped");
    break;
default:
    break;
}
}

esp_event_loop_create_default(); // 创建一个在后台运行的默认事件循环
esp_event_handler_register(ETH_EVENT, ESP_EVENT_ANY_ID, &eth_event_handler, NULL); // 注册以太网事件处理程序（用于在发生 link up/down 等事件时，处理特定的用户相关内容）

启动以太网驱动程序 安装驱动程序后，可以立即启动以太网。

esp_eth_start(eth_handle); // 启动以太网驱动程序状态机

连接驱动程序至 TCP/IP 协议栈 现在，以太网驱动程序已经完成安装。但对应 OSI（开放系统互连模型）来看，目前阶段仍然属于第二层（即数据链路层）。这意味着可以检测到 link up/down 事件，获得用户空间的 MAC 地址，但无法获得 IP 地址，当然也无法发送 HTTP 请求。ESP-IDF 中使用的 TCP/IP 协议栈是 LwIP，关于 LwIP 的更多信息，请参考 LwIP。

要将以太网驱动程序连接到 TCP/IP 协议栈，需要以下三步：
1. 为以太网驱动程序创建网络接口
2. 将网络接口连接到以太网驱动程序
3. 注册 IP 事件处理程序

有关网络接口的更多信息，请参考 Network Interface。

/** IP_EVENT_ETH_GOT_IP 的事件处理程序 */
static void got_ip_event_handler(void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)
{
    ip_event_got_ip_t *event = (ip_event_got_ip_t *) event_data;
    const esp_netif_ip_info_t *ip_info = &event->ip_info;

    ESP_LOGI(TAG, "Ethernet Got IP Address");
    ESP_LOGI(TAG, "~~~~~~~~~~");
    ESP_LOGI(TAG, "ETHIP: %s", IPSTR(&ip_info->ip));
    ESP_LOGI(TAG, "ETHMASK: %s", IPSTR(&ip_info->netmask));
    ESP_LOGI(TAG, "ETHGW: %s", IPSTR(&ip_info->gw));
    ESP_LOGI(TAG, "~~~~~~~~~~");
}

esp_netif_init(); // 初始化 TCP/IP 网络接口（在应用程序中应仅调用一次）
esp_netif_config_t cfg = ESP_NETIF_DEFAULT_ETH(); // 应用以太网的默认网络接口配置
esp_netif_t *eth_netif = esp_netif_new(cfg); // 为以太网驱动程序创建网络接口
esp_netif_attach(eth_netif, esp_eth_new_netif_glue(eth_handle)); //...

-- 将以太网驱动程序连接到 TCP/IP 协议栈
esp_event_handler_register(IP_EVENT, IP_EVENT_ETH_GOT_IP, got_ip_event_handler, NULL); // 注册用户定义的 IP 事件处理程序

(下页继续)
esp_eth_start(eth_handle);  // 启动以太网驱动程序状态机

警告：推荐在完成整个以太网驱动和网络接口的初始化后，再注册用户定义的以太网/IP 事件处理程序。也就是把注册事件处理程序作为启动以太网驱动程序的最后一步。这样可以确保以太网驱动程序或网络接口将首先执行以太网/IP 事件，从而保证在执行用户定义的处理程序时，系统处于预期状态。

以太网驱动程序的杂项控制：以下功能只支持在安装以太网驱动程序后调用。

- 关闭以太网驱动程序： esp_eth_stop()
- 更新以太网数据输入路径： esp_eth_update_input_path()
- 获取/设置以太网驱动程序简要内容： esp_eth_ioctl()

```
/* 获取 MAC 地址 */
uint8_t mac_addr[6];
memset(mac_addr, 0, sizeof(mac_addr));
esp_eth_ioctl(eth_handle, ETH_CMD_G_MAC_ADDR, mac_addr);
ESP_LOGI(TAG, "Ethernet MAC Address: %02x:%02x:%02x:%02x:%02x:%02x",
        mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_
        addr[5]);

/* 获取 PHY 地址 */
int phy_addr = -1;
esp_eth_ioctl(eth_handle, ETH_CMD_G_PHY_ADDR, &phy_addr);
ESP_LOGI(TAG, "Ethernet PHY Address: %d", phy_addr);
```

数据流量控制：受 RAM 大小限制，在网络拥挤时，MCU 上的以太网通常仅能处理有限数量的帧。发送站的数据传输速度可能快于对等端的接收能力。以太网数据流量控制机制允许接收节点向发送方发出信号，要求暂停传输，直到接收方跟上。这项功能是通过暂停帧实现的，该帧定义在 IEEE 802.3x 中。

暂停帧是一种特殊的以太帧，用于携带暂停命令，其 EtherType 字段为 0x8808，控制操作码为 0x0001。只有配置为全双工操作的节点组可以发送暂停帧。当节点组希望暂停链路的另一端时，它可以发送一个暂停帧到 48 位的保留组播地址 01-80-C2-00-00-01。暂停帧中也包括请求暂停的时间段，以两字节的整数形式发送，值的范围从 0 到 65535。

安装以太网驱动程序后，数据流量控制功能默认禁用，可以通过以下方式启用此功能：

```
bool flow_ctrl_enable = true;
esp_eth_ioctl(eth_handle, ETH_CMD_S_FLOW_CTRL, &flow_ctrl_enable);
```

需注意，暂停帧是在自协商期间由 PHY 向对等端公布的。只有当链路的两边都支持暂停帧时，以太网驱动程序才会发送暂停帧。

应用示例：
- 以太网基本示例： ethernet/basic
- 以太网 iperf 示例： ethernet/iperf
- 以太网到 Wi-Fi AP “路由器”： ethernet/eth2ap
- 大多数协议示例也适用于以太网： protocols

进阶操作：

自定义 PHY 驱动程序：目前市面上已有多家 PHY 制造商提供了大量的芯片组合。ESP-IDF 现已支持数种 PHY 芯片，但是由于价格、功能、库存等原因，有时用户还是无法找到一款能满足其实际需求的芯片。
Chapter 2. API 参考

好在 IEEE 802.3 在其 22.2.4 管理功能部分对 EMAC 和 PHY 之间的管理接口进行了标准化。该部分定义了所谓的” MII 管理接口”规范，用于控制 PHY 和收集 PHY 的状态，还定义了一组管理寄存器控制芯片行为、链接属性、自动协商配置等。在 ESP-IDF 中，这项基本的管理功能是由 esp_eth/src/esp_eth_phy_802_3.c 实现的，这也大大降低了创建新的自定义 PHY 芯片驱动的难度。

备注：由于一些 PHY 芯片可能不符合 IEEE 802.3 第 22.2.4 节的规定，所以请首先查看 PHY 数据手册。不过，就算芯片不符合规定，您依旧可以创建自定义 PHY 驱动程序，只是由于需要自行定义所有的 PHY 管理功能，这个过程将变得较为复杂。

ESP-IDF 以太网驱动程序所需的大部分 PHY 管理功能都已涵盖在 esp_eth/src/esp_eth_phy_802_3.c 中。不过对于以下几项，可能仍需针对不同芯片开发具体的管理功能：

- 链接状态。此项总使用以使用的具体芯片决定
- 芯片初始化。即使不存在严格的限制，也应进行自定义，以确保使用的是符合预期的芯片
- 芯片的具体功能配置

创建自定义 PHY 驱动程序的步骤：

1. 请根据 PHY 数据手册，定义针对供应商的特定注册布局。示例如参见 esp_eth/src/esp_eth_phy_ip101.c。
2. 准备衍生的 PHY 管理对象信息结构，该结构：
   - 必须至少包含 IEEE 802.3 phy_802_3_t 父对象
   - 可选包含支持非 IEEE 802.3 或自定义功能所需的额外变量。示例如参见 esp_eth/src/esp_eth_phy_ksz80xx.c。
3. 定义针对芯片的特定管理回调功能。
4. 初始化 IEEE 802.3 父对象并重新分配针对芯片的特定管理回调功能。

实现新的自定义 PHY 驱动程序后，你可以通过 IDF 组件管理器将驱动分享给其他用户。

API 参考

Header File

- components/esp_eth/include/esp_eth.h

Header File

- components/esp_eth/include/esp_eth_driver.h

Functions

- esp_err_t esp_eth_driver_install (const esp_eth_config_t *config, esp_eth_handle_t *out_hdl)
  - 安装以太网驱动。
  - 参数
    - config  - [in] 配置的以太网驱动
    - out_hdl - [out] 以太网驱动句柄
  - 返回
    - ESP_OK: 安装以太网驱动成功
    - ESP_ERR_INVALID_ARG: 安装以太网驱动失败，因为有无效参数
    - ESP_ERR_NO_MEM: 安装以太网驱动失败，因为没有内存
    - ESP_FAIL: 安装以太网驱动失败，因为其他错误发生

- esp_err_t esp_eth_driver_uninstall (esp_eth_handle_t hdld)
  - 卸载以太网驱动。
Chapter 2. API 参考

It’s not recommended to uninstall Ethernet driver unless it won’t get used any more in application code. To uninstall Ethernet driver, you have to make sure, all references to the driver are released. Ethernet driver can only be uninstalled successfully when reference counter equals to one.

参数 `hdl` –[in] handle of Ethernet driver

返回
- ESP_OK: uninstall esp_eth driver successfully
- ESP_ERR_INVALID_ARG: uninstall esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: uninstall esp_eth driver failed because it has more than one reference
- ESP_FAIL: uninstall esp_eth driver failed because some other error occurred

```c
esp_err_t esp_eth_start (esp_eth_handle_t hdl)
```
Start Ethernet driver ONLY in standalone mode (i.e. without TCP/IP stack)

备注： This API will start driver state machine and internal software timer (for checking link status).

参数 `hdl` –[in] handle of Ethernet driver

返回
- ESP_OK: start esp_eth driver successfully
- ESP_ERR_INVALID_ARG: start esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: start esp_eth driver failed because driver has started already
- ESP_FAIL: start esp_eth driver failed because some other error occurred

```c
esp_err_t esp_eth_stop (esp_eth_handle_t hdl)
```
Stop Ethernet driver.

备注： This function does the opposite operation of `esp_eth_start`.

参数 `hdl` –[in] handle of Ethernet driver

返回
- ESP_OK: stop esp_eth driver successfully
- ESP_ERR_INVALID_ARG: stop esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: stop esp_eth driver failed because driver has not started yet
- ESP_FAIL: stop esp_eth driver failed because some other error occurred

```c
esp_err_t esp_eth_update_input_path (esp_eth_handle_t hdl, esp_err_t (*stack_input)(esp_eth_handle_t hdl, uint8_t *buffer, uint32_t length, void *priv), void *priv)
```
Update Ethernet data input path (i.e. specify where to pass the input buffer)

备注： After install driver, Ethernet still don’t know where to deliver the input buffer. In fact, this API registers a callback function which get invoked when Ethernet received new packets.

参数
- `hdl` –[in] handle of Ethernet driver
- `stack_input` –[in] function pointer, which does the actual process on incoming packets
Chapter 2. API

• **priv** - [in] private resource, which gets passed to `stack_input` callback without any modification
  
  返回
  • ESP_OK: update input path successfully
  • ESP_ERR_INVALID_ARG: update input path failed because of some invalid argument
  • ESP_FAIL: update input path failed because some other error occurred

`esp_err_t esp_eth_transmit(esp_eth_handle_t hdl, void *buf, size_t length)`

General Transmit.

参数
  • hdl - [in] handle of Ethernet driver
  • buf - [in] buffer of the packet to transfer
  • length - [in] length of the buffer to transfer
  
  返回
  • ESP_OK: transmit frame buffer successfully
  • ESP_ERR_INVALID_ARG: transmit frame buffer failed because of some invalid argument
  • ESP_ERR_INVALID_STATE: invalid driver state (e.i. driver is not started)
  • ESP_ERR_TIMEOUT: transmit frame buffer failed because HW was not get available in predefined period
  • ESP_FAIL: transmit frame buffer failed because some other error occurred

`esp_err_t esp_eth_transmit_vargs(esp_eth_handle_t hdl, uint32_t argc, ...)`

Special Transmit with variable number of arguments.

参数
  • hdl - [in] handle of Ethernet driver
  • argc - [in] number variable arguments
  • ... - variable arguments
  
  返回
  • ESP_OK: transmit successfull
  • ESP_ERR_INVALID_STATE: invalid driver state (e.i. driver is not started)
  • ESP_ERR_TIMEOUT: transmit frame buffer failed because HW was not get available in predefined period
  • ESP_FAIL: transmit frame buffer failed because some other error occurred

`esp_err_t esp_eth_ioctl(esp_eth_handle_t hdl, esp_eth_io_cmd_t cmd, void *data)`

Misc IO function of Ethernet driver.

The following common IO control commands are supported:

• **ETH_CMD_S_MAC_ADDR** sets Ethernet interface MAC address. `data` argument is pointer to MAC address buffer with expected size of 6 bytes.

• **ETH_CMD_G_MAC_ADDR** gets Ethernet interface MAC address. `data` argument is pointer to a buffer to which MAC address is to be copied. The buffer size must be at least 6 bytes.

• **ETH_CMD_S_PHY_ADDR** sets PHY address in range of <0-31>. `data` argument is pointer to memory of `uint32_t` datatype from where the configuration option is read.

• **ETH_CMD_G_PHY_ADDR** gets PHY address. `data` argument is pointer to memory of `uint32_t` datatype to which the PHY address is to be stored.

• **ETH_CMD_S_AUTONEGO** enables or disables Ethernet link speed and duplex mode autonegotiation. `data` argument is pointer to memory of `bool` datatype from which the configuration option is read. Preconditions: Ethernet driver needs to be stopped.

• **ETH_CMD_G_AUTONEGO** gets current configuration of the Ethernet link speed and duplex mode autonegotiation. `data` argument is pointer to memory of `bool` datatype to which the current configuration is to be stored.

• **ETH_CMD_S_SPEED** sets the Ethernet link speed. `data` argument is pointer to memory of `eth_speed_t` datatype from which the configuration option is read. Preconditions: Ethernet driver needs to be stopped and auto-negotiation disabled.
Chapter 2. API 参考

- **ETH_CMD_G_SPEED** gets current Ethernet link speed. data argument is pointer to memory of eth_speed_t datatype to which the speed is to be stored.
- **ETH_CMD_S_PROMISCUOUS** sets/resets Ethernet interface promiscuous mode. data argument is pointer to memory of bool datatype from which the configuration option is read.
- **ETH_CMD_S_FLOW_CTRL** sets/resets Ethernet interface flow control. data argument is pointer to memory of bool datatype from which the configuration option is read.
- **ETH_CMD_S_DUPLEX_MODE** sets the Ethernet duplex mode. data argument is pointer to memory of eth_duplex_t datatype from which the configuration option is read. Preconditions: Ethernet driver needs to be stopped and auto-negotiation disabled.
- **ETH_CMD_G_DUPLEX_MODE** gets current Ethernet link duplex mode. data argument is pointer to memory of eth_duplex_t datatype to which the duplex mode is to be stored.
- **ETH_CMD_S_PHY_LOOPBACK** sets/resets PHY to/from loopback mode. data argument is pointer to memory of bool datatype from which the configuration option is read.
- Note that additional control commands may be available for specific MAC or PHY chips. Please consult specific MAC or PHY documentation or driver code.

### Parameters
- **hdl** - [in] handle of Ethernet driver
- **cmd** - [in] IO control command
- **data** - [inout] address of data for set command or address where to store the data when used with get command

### Return
- **ESP_OK**: process io command successfully
- **ESP_ERR_INVALID_ARG**: process io command failed because of some invalid argument
- **ESP_FAIL**: process io command failed because some other error occurred
- **ESP_ERR_NOT_SUPPORTED**: requested feature is not supported

```c
esp_err_t esp_eth_increase_reference(esp_eth_handle_t hdl)
```
Increase Ethernet driver reference.

备注：Ethernet driver handle can be obtained by os timer, netif, etc. It’s dangerous when thread A is using Ethernet but thread B uninstall the driver. Using reference counter can prevent such risk, but care should be taken, when you obtain Ethernet driver; this API must be invoked so that the driver won’t be uninstalled during your using time.

```c
esp_err_t esp_eth_decrease_reference(esp_eth_handle_t hdl)
```
Decrease Ethernet driver reference.

### Structures

```c
struct esp_eth_config_t
```
Configuration of Ethernet driver.
Public Members

`esp_eth_mac_t *mac`
Ethernet MAC object.

`esp_eth_phy_t *phy`
Ethernet PHY object.

`uint32_t check_link_period_ms`
Period time of checking Ethernet link status.

`esp_err_t (*stack_input)(esp_eth_handle_t eth_handle, uint8_t*buffer, uint32_t length, void *priv)`
Input frame buffer to user’s stack.

- **Param eth_handle [in]** handle of Ethernet driver
- **Param buffer [in]** frame buffer that will get input to upper stack
- **Param length [in]** length of the frame buffer
- **Return**
  - ESP_OK: input frame buffer to upper stack successfully
  - ESP_FAIL: error occurred when inputting buffer to upper stack

`esp_err_t (*on_lowlevel_init_done)(esp_eth_handle_t eth_handle)`
Callback function invoked when lowlevel initialization is finished.

- **Param eth_handle [in]** handle of Ethernet driver
- **Return**
  - ESP_OK: process extra lowlevel initialization successfully
  - ESP_FAIL: error occurred when processing extra lowlevel initialization

`esp_err_t (*on_lowlevel_deinit_done)(esp_eth_handle_t eth_handle)`
Callback function invoked when lowlevel deinitialization is finished.

- **Param eth_handle [in]** handle of Ethernet driver
- **Return**
  - ESP_OK: process extra lowlevel deinitialization successfully
  - ESP_FAIL: error occurred when processing extra lowlevel deinitialization

`esp_err_t (*read_phy_reg)(esp_eth_handle_t eth_handle, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)`
Read PHY register.

- **Param eth_handle [in]** handle of Ethernet driver
- **Param phy_addr [in]** PHY chip address (0-31)
- **Param phy_reg [in]** PHY register index code
- **Param reg_value [out]** PHY register value
- **Return**
  - ESP_OK: read PHY register successfully
  - ESP_ERR_INVALID_ARG: read PHY register failed because of invalid argument
  - ESP_ERR_TIMEOUT: read PHY register failed because of timeout
  - ESP_FAIL: read PHY register failed because some other error occurred

**Note:** Usually the PHY register read/write function is provided by MAC (SMI interface), but if the PHY device is managed by other interface (e.g. I2C), then user needs to implement the corresponding read/write. Setting this to NULL means your PHY device is managed by MAC’s SMI interface.
`esp_err_t (*write_phy_reg)(esp_eth_handle_t eth_handle, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)`

Write PHY register.

**Param**
- `eth_handle` [in] handle of Ethernet driver
- `phy_addr` [in] PHY chip address (0-31)
- `phy_reg` [in] PHY register index code
- `reg_value` [in] PHY register value

**Return**
- ESP_OK: write PHY register successfully
- ESP_ERR_INVALID_ARG: read PHY register failed because of invalid argument
- ESP_ERR_TIMEOUT: write PHY register failed because of timeout
- ESP_FAIL: write PHY register failed because some other error occurred

**Macros**

```c
ETH_DEFAULT_CONFIG(emac, ephy)
```

Default configuration for Ethernet driver.

**Type Definitions**

```c
typedef void *esp_eth_handle_t
```

Handle of Ethernet driver.

**Enumerations**

```c
typedef enum esp_eth_io_cmd_t
```

Command list for ioctl API.

**Values:**

```c
enumerator ETH_CMD_G_MAC_ADDR
        Get MAC address
```

```c
enumerator ETH_CMD_S_MAC_ADDR
        Set MAC address
```

```c
enumerator ETH_CMD_G_PHY_ADDR
        Get PHY address
```

```c
enumerator ETH_CMD_S_PHY_ADDR
        Set PHY address
```

```c
enumerator ETH_CMD_G_AUTONEGO
        Get PHY Auto Negotiation
```

```c
enumerator ETH_CMD_S_AUTONEGO
        Set PHY Auto Negotiation
```

备注: Usually the PHY register read/write function is provided by MAC (SMI interface), but if the PHY device is managed by other interface (e.g. I2C), then user needs to implement the corresponding read/write. Setting this to NULL means your PHY device is managed by MAC’s SMI interface.
enumerator **ETH_CMD_G_SPEED**
   Get Speed

enumerator **ETH_CMD_S_SPEED**
   Set Speed

enumerator **ETH_CMD_S_PROMISCUOUS**
   Set promiscuous mode

enumerator **ETH_CMD_S_FLOW_CTRL**
   Set flow control

enumerator **ETH_CMD_G_DUPLEX_MODE**
   Get Duplex mode

enumerator **ETH_CMD_S_DUPLEX_MODE**
   Set Duplex mode

enumerator **ETH_CMD_S_PHY_LOOPBACK**
   Set PHY loopback

enumerator **ETH_CMD_CUSTOM_MAC_CMDS**

enumerator **ETH_CMD_CUSTOM_PHY_CMDS**

**Header File**

- components/esp_eth/include/esp_eth_com.h

**Structures**

struct **esp_eth_mediator_s**

   Ethernet mediator.

**Public Members**

```c
esp_err_t (*phy_reg_read)(esp_eth_mediator_t *eth, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)
```

Read PHY register.

- **Param eth** [in] mediator of Ethernet driver
- **Param phy_addr** [in] PHY Chip address (0~31)
- **Param phy_reg** [in] PHY register index code
- **Param reg_value** [out] PHY register value

**Return**

- ESP_OK: read PHY register successfully
- ESP_FAIL: read PHY register failed because some error occurred
**esp_err_t (**phy_reg_write**)(**esp_eth_mediator_t** *eth, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)**

Write PHY register.

- **Param eth** [in] mediator of Ethernet driver
- **Param phy_addr** [in] PHY Chip address (0~31)
- **Param phy_reg** [in] PHY register index code
- **Param reg_value** [in] PHY register value

**Return**
- ESP_OK: write PHY register successfully
- ESP_FAIL: write PHY register failed because some error occurred

**esp_err_t (**stack_input**)(**esp_eth_mediator_t** *eth, uint8_t *buffer, uint32_t length)**

Deliver packet to upper stack.

- **Param eth** [in] mediator of Ethernet driver
- **Param buffer** [in] packet buffer
- **Param length** [in] length of the packet

**Return**
- ESP_OK: deliver packet to upper stack successfully
- ESP_FAIL: deliver packet failed because some error occurred

**esp_err_t (**on_state_changed**)(**esp_eth_mediator_t** *eth, **esp_eth_state_t** state, void *args)**

Callback on Ethernet state changed.

- **Param eth** [in] mediator of Ethernet driver
- **Param state** [in] new state
- **Param args** [in] optional argument for the new state

**Return**
- ESP_OK: process the new state successfully
- ESP_FAIL: process the new state failed because some error occurred

### Type Definitions

typedef struct **esp_eth_mediator_s** *esp_eth_mediator_t

Ethernet mediator.

### Enumerations

enum **esp_eth_state_t**

Ethernet driver state.

**Values:**

- enumerator **ETH_STATE_LLINIT**
  Low level init done

- enumerator **ETH_STATE_DEINIT**
  Deinit done

- enumerator **ETH_STATE_LINK**
  Link status changed

- enumerator **ETH_STATE_SPEED**
  Speed updated
enumerator ETH_STATE_DUPLEX
    Duplex updated

eenumerator ETH_STATE_PAUSE
    Pause ability updated

eenum eth_event_t
    Ethernet event declarations.
    Values:

eenumerator ETHERNET_EVENT_START
    Ethernet driver start

eenumerator ETHERNET_EVENT_STOP
    Ethernet driver stop

eenumerator ETHERNET_EVENT_CONNECTED
    Ethernet got a valid link

eenumerator ETHERNET_EVENT_DISCONNECTED
    Ethernet lost a valid link

**Header File**

- components/esp_eth/include/esp_eth_mac.h

**Functions**

*esp_eth_mac_t* esp_eth_mac_new_esp32 (const *eth_esp32_emac_config_t* esp32_config, const *eth_mac_config_t* config)

Create ESP32 Ethernet MAC instance.

**参数**

- esp32_config - EMAC specific configuration
- config - Ethernet MAC configuration

**返回**

- instance: create MAC instance successfully
- NULL: create MAC instance failed because some error occurred

**Unions**

union eth_mac_clock_config_t
    #include <esp_eth_mac.h> Ethernet MAC Clock Configuration.

**Public Members**

struct eth_mac_clock_config_t::[anonymous] mii
    EMAC MII Clock Configuration

emac_rmii_clock_mode_t clock_mode
    RMII Clock Mode Configuration
**Chapter 2. API**

```c
emac_rmii_clock_gpio_t clock_gpio
```
RMII Clock GPIO Configuration

```c
struct eth_mac_clock_config_t [anonymous] rmii
```
EMAC RMII Clock Configuration

### Structures

```c
struct esp_eth_mac_s
```
Ethernet MAC.

#### Public Members

```c
esp_err_t (*set_mediator)(esp_eth_mac_t *mac, esp_eth_mediator_t *eth)
```
Set mediator for Ethernet MAC.

- **Param** `mac [in]` Ethernet MAC instance
- **Param** `eth [in]` Ethernet mediator

**Return**
- ESP_OK: set mediator for Ethernet MAC successfully
- ESP_ERR_INVALID_ARG: set mediator for Ethernet MAC failed because of invalid argument

```c
esp_err_t (*init)(esp_eth_mac_t *mac)
```
Initialize Ethernet MAC.

- **Param** `mac [in]` Ethernet MAC instance

**Return**
- ESP_OK: initialize Ethernet MAC successfully
- ESP_ERR_TIMEOUT: initialize Ethernet MAC failed because of timeout
- ESP_FAIL: initialize Ethernet MAC failed because some other error occurred

```c
esp_err_t (*deinit)(esp_eth_mac_t *mac)
```
Deinitialize Ethernet MAC.

- **Param** `mac [in]` Ethernet MAC instance

**Return**
- ESP_OK: deinitialize Ethernet MAC successfully
- ESP_FAIL: deinitialize Ethernet MAC failed because some other error occurred

```c
esp_err_t (*start)(esp_eth_mac_t *mac)
```
Start Ethernet MAC.

- **Param** `mac [in]` Ethernet MAC instance

**Return**
- ESP_OK: start Ethernet MAC successfully
- ESP_FAIL: start Ethernet MAC failed because some other error occurred

```c
esp_err_t (*stop)(esp_eth_mac_t *mac)
```
Stop Ethernet MAC.

- **Param** `mac [in]` Ethernet MAC instance

**Return**
- ESP_OK: stop Ethernet MAC successfully
• **ESP_FAIL**: stop Ethernet MAC failed because some error occurred

```
esp_err_t (*transmit)(esp_eth_mac_t *mac, uint8_t *buf, uint32_t length)
```

Transmit packet from Ethernet MAC.

<table>
<thead>
<tr>
<th>Param</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac</td>
<td>[in]</td>
<td>Ethernet MAC instance</td>
</tr>
<tr>
<td>buf</td>
<td>[in]</td>
<td>packet buffer to transmit</td>
</tr>
<tr>
<td>length</td>
<td>[in]</td>
<td>length of packet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESP_OK</td>
<td>transmit packet successfully</td>
</tr>
<tr>
<td></td>
<td>ESP_ERR_INVALID_SIZE</td>
<td>number of actually sent bytes differs to expected</td>
</tr>
<tr>
<td></td>
<td>ESP_FAIL</td>
<td>transmit packet failed because some other error occurred</td>
</tr>
</tbody>
</table>

```
esp_err_t (*transmit_vargs)(esp_eth_mac_t *mac, uint32_t argc, va_list args)
```

Transmit packet from Ethernet MAC constructed with special parameters at Layer2.

<table>
<thead>
<tr>
<th>Param</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac</td>
<td>[in]</td>
<td>Ethernet MAC instance</td>
</tr>
<tr>
<td>argc</td>
<td>[in]</td>
<td>num variable arguments</td>
</tr>
<tr>
<td>args</td>
<td>[in]</td>
<td>variable arguments</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESP_OK</td>
<td>transmit packet successfully</td>
</tr>
<tr>
<td></td>
<td>ESP_ERR_INVALID_SIZE</td>
<td>number of actually sent bytes differs to expected</td>
</tr>
<tr>
<td></td>
<td>ESP_FAIL</td>
<td>transmit packet failed because some other error occurred</td>
</tr>
</tbody>
</table>

```
esp_err_t (*receive)(esp_eth_mac_t *mac, uint8_t *buf, uint32_t *length)
```

Receive packet from Ethernet MAC.

<table>
<thead>
<tr>
<th>Param</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac</td>
<td>[in]</td>
<td>Ethernet MAC instance</td>
</tr>
<tr>
<td>buf</td>
<td>[out]</td>
<td>packet buffer which will preserve the received frame</td>
</tr>
<tr>
<td>length</td>
<td>[out]</td>
<td>length of the received packet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESP_OK</td>
<td>receive packet successfully</td>
</tr>
<tr>
<td></td>
<td>ESP_ERR_INVALID_ARG</td>
<td>receive packet failed because of invalid argument</td>
</tr>
<tr>
<td></td>
<td>ESP_ERR_INVALID_SIZE</td>
<td>input buffer size is not enough to hold the incoming data. in this case, value of returned “length” indicates the real size of incoming data.</td>
</tr>
<tr>
<td></td>
<td>ESP_FAIL</td>
<td>receive packet failed because some other error occurred</td>
</tr>
</tbody>
</table>

备注: Returned error codes may differ for each specific MAC chip.

备注: Typical intended use case is to make possible to construct a frame from multiple higher layer buffers without a need of buffer reallocations. However, other use cases are not limited.

备注: Memory of buf is allocated in the Layer2, make sure it get free after process.

备注: Before this function got invoked, the value of “length” should set by user, equals the size of buffer. After the function returned, the value of “length” means the real length of received data.
### Chapter 2. API

**esp_err_t (*read_phy_reg)(esp_eth_mac_t *mac, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)**

Read PHY register.

**Param**

- **mac** [in] Ethernet MAC instance
- **phy_addr** [in] PHY chip address (0~31)
- **phy_reg** [in] PHY register index code
- **reg_value** [out] PHY register value

**Return**

- **ESP_OK**: read PHY register successfully
- **ESP_ERR_INVALID_ARG**: read PHY register failed because of invalid argument
- **ESP_ERR_INVALID_STATE**: read PHY register failed because of wrong state of MAC
- **ESP_ERR_TIMEOUT**: read PHY register failed because of timeout
- **ESP_FAIL**: read PHY register failed because some other error occurred

**esp_err_t (*write_phy_reg)(esp_eth_mac_t *mac, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)**

Write PHY register.

**Param**

- **mac** [in] Ethernet MAC instance
- **phy_addr** [in] PHY chip address (0~31)
- **phy_reg** [in] PHY register index code
- **reg_value** [in] PHY register value

**Return**

- **ESP_OK**: write PHY register successfully
- **ESP_ERR_INVALID_STATE**: write PHY register failed because of wrong state of MAC
- **ESP_ERR_TIMEOUT**: write PHY register failed because of timeout
- **ESP_FAIL**: write PHY register failed because some other error occurred

**esp_err_t (*set_addr)(esp_eth_mac_t *mac, uint8_t *addr)**

Set MAC address.

**Param**

- **mac** [in] Ethernet MAC instance
- **addr** [in] MAC address

**Return**

- **ESP_OK**: set MAC address successfully
- **ESP_ERR_INVALID_ARG**: set MAC address failed because of invalid argument
- **ESP_FAIL**: set MAC address failed because some other error occurred

**esp_err_t (*get_addr)(esp_eth_mac_t *mac, uint8_t *addr)**

Get MAC address.

**Param**

- **mac** [in] Ethernet MAC instance
- **addr** [out] MAC address

**Return**

- **ESP_OK**: get MAC address successfully
- **ESP_ERR_INVALID_ARG**: get MAC address failed because of invalid argument
- **ESP_FAIL**: get MAC address failed because some other error occurred

**esp_err_t (*set_speed)(esp_eth_mac_t *mac, eth_speed_t speed)**

Set speed of MAC.

**Param**

- **mac** [in] Ethernet MAC instance
- **speed** [in] MAC speed

**Return**

- **ESP_OK**: set MAC speed successfully
• ESP_ERR_INVALID_ARG: set MAC speed failed because of invalid argument
• ESP_FAIL: set MAC speed failed because some other error occurred

```
esp_err_t (*set_duplex)(esp_eth_mac_t *mac, eth_duplex_t duplex)
```

Set duplex mode of MAC.

**Param**
- mac [in] Ethernet MAC instance
- duplex [in] MAC duplex

**Return**
- ESP_OK: set MAC duplex mode successfully
- ESP_ERR_INVALID_ARG: set MAC duplex failed because of invalid argument
- ESP_FAIL: set MAC duplex failed because some other error occurred

```
esp_err_t (*set_link)(esp_eth_mac_t *mac, eth_link_t link)
```

Set link status of MAC.

**Param**
- mac [in] Ethernet MAC instance
- link [in] Link status

**Return**
- ESP_OK: set link status successfully
- ESP_ERR_INVALID_ARG: set link status failed because of invalid argument
- ESP_FAIL: set link status failed because some other error occurred

```
esp_err_t (*set_promiscuous)(esp_eth_mac_t *mac, bool enable)
```

Set promiscuous of MAC.

**Param**
- mac [in] Ethernet MAC instance
- enable [in] set true to enable promiscuous mode; set false to disable promiscuous mode

**Return**
- ESP_OK: set promiscuous mode successfully
- ESP_FAIL: set promiscuous mode failed because some error occurred

```
esp_err_t (*enable_flow_ctrl)(esp_eth_mac_t *mac, bool enable)
```

Enable flow control on MAC layer or not.

**Param**
- mac [in] Ethernet MAC instance
- enable [in] set true to enable flow control; set false to disable flow control

**Return**
- ESP_OK: set flow control successfully
- ESP_FAIL: set flow control failed because some error occurred

```
esp_err_t (*set_peer_pause_ability)(esp_eth_mac_t *mac, uint32_t ability)
```

Set the PAUSE ability of peer node.

**Param**
- mac [in] Ethernet MAC instance
- ability [in] zero indicates that pause function is supported by link partner; non-zero indicates that pause function is not supported by link partner

**Return**
- ESP_OK: set peer pause ability successfully
- ESP_FAIL: set peer pause ability failed because some other error occurred

```
esp_err_t (*custom_ioctl)(esp_eth_mac_t *mac, uint32_t cmd, void *data)
```

Custom IO function of MAC driver. This function is intended to extend common options of esp_eth_ioctl to cover specifics of MAC chip.

---

**Note:** This function may not be assigned when the MAC chip supports only most common set of
configuration options.

**Param** mac [in] Ethernet MAC instance

**Param** cmd [in] IO control command

**Param** data [inout] address of data for set command or address where to store the data when used with get command

**Return**
- ESP_OK: process io command successfully
- ESP_ERR_INVALID_ARG: process io command failed because of some invalid argument
- ESP_FAIL: process io command failed because some other error occurred
- ESP_ERR_NOT_SUPPORTED: requested feature is not supported

`esp_err_t (*del)(esp_eth_mac_t *mac)`

Free memory of Ethernet MAC.

**Param** mac [in] Ethernet MAC instance

**Return**
- ESP_OK: free Ethernet MAC instance successfully
- ESP_FAIL: free Ethernet MAC instance failed because some error occurred

```
struct eth_mac_config_t

Configuration of Ethernet MAC object.

**Public Members**

`uint32_t sw_reset_timeout_ms`

Software reset timeout value (Unit: ms)

`uint32_t rx_task_stack_size`

Stack size of the receive task

`uint32_t rx_task_prio`

Priority of the receive task

`uint32_t flags`

Flags that specify extra capability for mac driver
```

```

```

```

`struct eth_esp32_emac_config_t`

EMAC specific configuration.

**Public Members**

`int smi_mdc_gpio_num`

SMI MDC GPIO number, set to -1 could bypass the SMI GPIO configuration

`int smi_mdio_gpio_num`

SMI MDIO GPIO number, set to -1 could bypass the SMI GPIO configuration
Chapter 2. API

**EMAC Data interface to PHY (MII/RMII)**

- **eth_data_interface_t**
  - Interface type for EMAC Data

- **eth_mac_clock_config_t**
  - Interface clock configuration type

- **eth_mac_dma_burst_len_t**
  - EMAC DMA burst length for both Tx and Rx

**Macros**

- **ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE**
  - MAC driver can work when cache is disabled

- **ETH_MAC_FLAG_PIN_TO_CORE**
  - Pin MAC task to the CPU core where driver installation happened

- **ETH_MAC_DEFAULT_CONFIG()**
  - Default configuration for Ethernet MAC object.

- **ETH_ESP32_EMAC_DEFAULT_CONFIG()**
  - Default ESP32’s EMAC specific configuration.

**Type Definitions**

```c
typedef struct esp_eth_mac_s esp_eth_mac_t
```

- **Ethernet MAC**

**Enumerations**

- **emac_rmii_clock_mode_t**
  - RMII Clock Mode Options.
  - **Values:**
    - **enumerator EMAC_CLK_DEFAULT**
      - Default values configured using Kconfig are going to be used when “Default” selected.
    - **enumerator EMAC_CLK_EXT_IN**
      - Input RMII Clock from external. EMAC Clock GPIO number needs to be configured when this option is selected.
      - **Notes:** MAC will get RMII clock from outside. Note that ESP32 only supports GPIO0 to input the RMII clock.
    - **enumerator EMAC_CLK_OUT**
      - Output RMII Clock from internal APLL Clock. EMAC Clock GPIO number needs to be configured when this option is selected.

- **emac_rmii_clock_gpio_t**
  - RMII Clock GPIO number Options.
  - **Values:**
enumerator **EMAC_CLK_IN_GPIO**
MAC will get RMII clock from outside at this GPIO.

备注：ESP32 only supports GPIO0 to input the RMII clock.

enumerator **EMAC_APPL_CLK_OUT_GPIO**
Output RMII Clock from internal APLL Clock available at GPIO0.

备注：GPIO0 can be set to output a pre-divided PLL clock (test only!). Enabling this option will configure GPIO0 to output a 50MHz clock. In fact this clock doesn’t have directly relationship with EMAC peripheral. Sometimes this clock won’t work well with your PHY chip. You might need to add some extra devices after GPIO0 (e.g. inverter). Note that outputting RMII clock on GPIO0 is an experimental practice. If you want the Ethernet to work with WiFi, don’t select GPIO0 output mode for stability.

enumerator **EMAC_CLK_OUT_GPIO**
Output RMII Clock from internal APLL Clock available at GPIO16.

enumerator **EMAC_CLK_OUT_180_GPIO**
Inverted Output RMII Clock from internal APLL Clock available at GPIO17.

**Header File**
- components/esp_eth/include/esp_eth_phy.h

**Functions**

```c
esp_eth_phy_t *esp_eth_phy_new_ip101 (const eth_phy_config_t *config)
Create a PHY instance of IP101.
```

参数 config [in] configuration of PHY

返回
- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

```c
esp_eth_phy_t *esp_eth_phy_new_rtl8201 (const eth_phy_config_t *config)
Create a PHY instance of RTL8201.
```

参数 config [in] configuration of PHY

返回
- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

```c
esp_eth_phy_t *esp_eth_phy_new_lan87xx (const eth_phy_config_t *config)
Create a PHY instance of LAN87xx.
```

参数 config [in] configuration of PHY

返回
- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

```c
esp_eth_phy_t *esp_eth_phy_new_dp83848 (const eth_phy_config_t *config)
Create a PHY instance of DP83848.
```

参数 config [in] configuration of PHY

返回
• instance: create PHY instance successfully
• NULL: create PHY instance failed because some error occurred

```c
esp_eth_phy_t *esp_eth Phy_new_ksz80xx(const eth Phy_config_t *config)
```

Create a PHY instance of KSZ80xx.

The phy model from the KSZ80xx series is detected automatically. If the driver is unable to detect a supported model, NULL is returned.

Currently, the following models are supported: KSZ8001, KSZ8021, KSZ8031, KSZ8041, KSZ8051, KSZ8061, KSZ8081, KSZ8091

参数 config [in] configuration of PHY
返回
• instance: create PHY instance successfully
• NULL: create PHY instance failed because some error occurred

**Structures**

```c
struct esp_eth_phy_s
```

Ethernet PHY.

**Public Members**

```c
esp_err_t (*set_mediator)(esp_eth_phy_t *phy, esp_eth_mediator_t *mediator)
```

Set mediator for PHY.

Param phy [in] Ethernet PHY instance
Param mediator [in] mediator of Ethernet driver
Return
• ESP_OK: set mediator for Ethernet PHY instance successfully
• ESP_ERR_INVALID_ARG: set mediator for Ethernet PHY instance failed because of some invalid arguments

```c
esp_err_t (*reset)(esp_eth_phy_t *phy)
```

Software Reset Ethernet PHY.

Param phy [in] Ethernet PHY instance
Return
• ESP_OK: reset Ethernet PHY successfully
• ESP_FAIL: reset Ethernet PHY failed because some error occurred

```c
esp_err_t (*reset_hw)(esp_eth_phy_t *phy)
```

Hardware Reset Ethernet PHY.

备注: Hardware reset is mostly done by pull down and up PHY’s nRST pin

```c
Param phy [in] Ethernet PHY instance
Return
• ESP_OK: reset Ethernet PHY successfully
• ESP_FAIL: reset Ethernet PHY failed because some error occurred

esp_err_t (*init)(esp_eth_phy_t *phy)
```

Initialize Ethernet PHY.
**Param phy** [in] Ethernet PHY instance

**Return**
- ESP_OK: initialize Ethernet PHY successfully
- ESP_FAIL: initialize Ethernet PHY failed because some error occurred

```c
esp_err_t (*deinit)(esp_eth_phy_t *phy)
```

Deinitialize Ethernet PHY.

**Param phy** [in] Ethernet PHY instance

**Return**
- ESP_OK: deinitialize Ethernet PHY successfully
- ESP_FAIL: deinitialize Ethernet PHY failed because some error occurred

```c
esp_err_t (*autonego_ctrl)(esp_eth_phy_t *phy, eth_phy_autoneg_cmd_t cmd, bool *autonego_en_stat)
```

Configure auto negotiation.

**Param phy** [in] Ethernet PHY instance

**Param cmd** [in] Configuration command, it is possible to Enable (restart), Disable or get current status of PHY auto negotiation

**Param autonego_en_stat** [out] Address where to store current status of auto negotiation configuration

**Return**
- ESP_OK: restart auto negotiation successfully
- ESP_FAIL: restart auto negotiation failed because some error occurred
- ESP_ERR_INVALID_ARG: invalid command

```c
esp_err_t (*get_link)(esp_eth_phy_t *phy)
```

Get Ethernet PHY link status.

**Param phy** [in] Ethernet PHY instance

**Return**
- ESP_OK: get Ethernet PHY link status successfully
- ESP_FAIL: get Ethernet PHY link status failed because some error occurred

```c
esp_err_t (*pwrctl)(esp_eth_phy_t *phy, bool enable)
```

Power control of Ethernet PHY.

**Param phy** [in] Ethernet PHY instance

**Param enable** [in] set true to power on Ethernet PHY; set false to power off Ethernet PHY

**Return**
- ESP_OK: control Ethernet PHY power successfully
- ESP_FAIL: control Ethernet PHY power failed because some error occurred

```c
esp_err_t (*set_addr)(esp_eth_phy_t *phy, uint32_t addr)
```

Set PHY chip address.

**Param phy** [in] Ethernet PHY instance

**Param addr** [in] PHY chip address

**Return**
- ESP_OK: set Ethernet PHY address successfully
- ESP_FAIL: set Ethernet PHY address failed because some error occurred

```c
esp_err_t (*get_addr)(esp_eth_phy_t *phy, uint32_t *addr)
```

Get PHY chip address.

**Param phy** [in] Ethernet PHY instance

**Param addr** [out] PHY chip address

**Return**
• ESP_OK: get Ethernet PHY address successfully
• ESP_ERR_INVALID_ARG: get Ethernet PHY address failed because of invalid argument

`esp_err_t (*advertise_pause_ability)(esp_eth_phy_t *phy, uint32_t ability)`
Advertise pause function supported by MAC layer.

Param `phy` [in] Ethernet PHY instance
Param `addr` [out] Pause ability
Return
• ESP_OK: Advertise pause ability successfully
• ESP_ERR_INVALID_ARG: Advertise pause ability failed because of invalid argument

`esp_err_t (*loopback)(esp_eth_phy_t *phy, bool enable)`
Sets the PHY to loopback mode.

Param `phy` [in] Ethernet PHY instance
Param `enable` [in] enables or disables PHY loopback
Return
• ESP_OK: PHY instance loopback mode has been configured successfully
• ESP_FAIL: PHY instance loopback configuration failed because some error occurred

`esp_err_t (*set_speed)(esp_eth_phy_t *phy, eth_speed_t speed)`
Sets PHY speed mode.

备注: Autonegotiation feature needs to be disabled prior to calling this function for the new setting to be applied

Param `phy` [in] Ethernet PHY instance
Param `speed` [in] Speed mode to be set
Return
• ESP_OK: PHY instance speed mode has been configured successfully
• ESP_FAIL: PHY instance speed mode configuration failed because some error occurred

`esp_err_t (*set_duplex)(esp_eth_phy_t *phy, eth_duplex_t duplex)`
Sets PHY duplex mode.

备注: Autonegotiation feature needs to be disabled prior to calling this function for the new setting to be applied

Param `phy` [in] Ethernet PHY instance
Param `duplex` [in] Duplex mode to be set
Return
• ESP_OK: PHY instance duplex mode has been configured successfully
• ESP_FAIL: PHY instance duplex mode configuration failed because some error occurred

`esp_err_t (*custom_ioctl)(esp_eth_phy_t *phy, uint32_t cmd, void *data)`
Custom IO function of PHY driver. This function is intended to extend common options of esp_eth_ioctl to cover specifics of PHY chip.
This function may not be assigned when the PHY chip supports only most common set of configuration options.

**Param**  
phy [in] Ethernet PHY instance  
**Param**  
cmd [in] IO control command  
**Param**  
data [inout] address of data for set command or address where to store the data when used with get command  

**Return**  
- ESP_OK: process io command successfully  
- ESP_ERR_INVALID_ARG: process io command failed because of some invalid argument  
- ESP_FAIL: process io command failed because some other error occurred  
- ESP_ERR_NOT_SUPPORTED: requested feature is not supported

```
esp_err_t (*del)(esp_eth_phy_t *phy)
```
Free memory of Ethernet PHY instance.

**Param**  
phy [in] Ethernet PHY instance  

**Return**  
- ESP_OK: free PHY instance successfully  
- ESP_FAIL: free PHY instance failed because some error occurred

```
struct eth_phy_config_t
```

Ethernet PHY configuration.

**Public Members**

```
int32_t phy_addr
```
PHY address, set -1 to enable PHY address detection at initialization stage

```
uint32_t reset_timeout_ms
```
Reset timeout value (Unit: ms)

```
uint32_t autonego_timeout_ms
```
Auto-negotiation timeout value (Unit: ms)

```
int reset_gpio_num
```
Reset GPIO number, -1 means no hardware reset

**Macros**

```
ESP_ETH_PHY_ADDR_AUTO
ETH_PHY_DEFAULT_CONFIG()
```
Default configuration for Ethernet PHY object.

**Type Definitions**

typedef struct esp_eth_phy_s esp_eth_phy_t
Ethernet PHY.
Enumerations

```c
enum eth_phy_autoneg_cmd_t
    Auto-negotiation control commands.
    Values:
```

enumerator ESP_ETH_PHY_AUTONEGO_RESTART
enumerator ESP_ETH_PHY_AUTONEGO_EN
enumerator ESP_ETH_PHY_AUTONEGO_DIS
enumerator ESP_ETH_PHY_AUTONEGO_G_STAT

Header File

- components/esp_eth/include/esp_eth_phy_802_3.h

Functions

```c
esp_err_t esp_eth_phy_802_3_reset_hw (phy_802_3_t *phy_802_3, uint32_t reset_assert_us)
    Performs hardware reset with specific reset pin assertion time.
```

参数

- phy_802_3 – IEEE 802.3 PHY object infrastructure
- reset_assert_us – Hardware reset pin assertion time

返回

- ESP_OK: reset Ethernet PHY successfully

```c
esp_err_t esp_eth_phy_802_3_detect_phy_addr (esp_eth_mediator_t *eth, int *detected_addr)
    Detect PHY address.
```

参数

- eth – Mediator of Ethernet driver
- detected_addr – [out] a valid address after detection

返回

- ESP_OK: detect phy address successfully
- ESP_ERR_INVALID_ARG: invalid parameter
- ESP_ERR_NOT_FOUND: can’t detect any PHY device
- ESP_FAIL: detect phy address failed because some error occurred

```c
esp_err_t esp_eth_phy_802_3_basic_phy_init (phy_802_3_t *phy_802_3)
    Performs basic PHY chip initialization.
```

备注：It should be called as the first function in PHY specific driver instance

```c
参数 phy_802_3 – IEEE 802.3 PHY object infrastructure
返回
- ESP_OK: initialized Ethernet PHY successfully
- ESP_FAIL: initialization of Ethernet PHY failed because some error occurred
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NOT_FOUND: PHY device not detected
- ESP_ERR_TIMEOUT: MII Management read/write operation timeout
- ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation
```


**Chapter 2. API**

*esp_err_t* `esp_eth_phy_802_3_basic_phy_deinit (phy_802_3_t *phy_802_3)`

Performs basic PHY chip de-initialization.

**备注:** It should be called as the last function in PHY specific driver instance

<table>
<thead>
<tr>
<th>参数</th>
<th>phy_802_3</th>
<th>IEEE 802.3 PHY object infostructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>返回</td>
<td></td>
<td>ESP_OK: de-initialized Ethernet PHY successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_FAIL: de-initialization of Ethernet PHY failed because some error occurred</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_TIMEOUT: MII Management read/write operation timeout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation</td>
</tr>
</tbody>
</table>

*esp_err_t* `esp_eth_phy_802_3_read_oui (phy_802_3_t *phy_802_3, uint32_t *oui)`

Reads raw content of OUI field.

<table>
<thead>
<tr>
<th>参数</th>
<th>phy_802_3</th>
<th>IEEE 802.3 PHY object infostructure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>oui [out]</td>
<td>OUI value</td>
</tr>
<tr>
<td>返回</td>
<td></td>
<td>ESP_OK: OUI field read successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_FAIL: OUI field read failed because some error occurred</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_INVALID_ARG: invalid oui argument</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_TIMEOUT: MII Management read/write operation timeout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation</td>
</tr>
</tbody>
</table>

*esp_err_t* `esp_eth_phy_802_3_read_manufac_info (phy_802_3_t *phy_802_3, uint8_t *model, uint8_t *rev)`

Reads manufacturer’s model and revision number.

<table>
<thead>
<tr>
<th>参数</th>
<th>phy_802_3</th>
<th>IEEE 802.3 PHY object infostructure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>model [out]</td>
<td>Manufacturer’s model number (can be NULL when not required)</td>
</tr>
<tr>
<td></td>
<td>rev [out]</td>
<td>Manufacturer’s revision number (can be NULL when not required)</td>
</tr>
<tr>
<td>返回</td>
<td></td>
<td>ESP_OK: Manufacturer’s info read successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_FAIL: Manufacturer’s info read failed because some error occurred</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_TIMEOUT: MII Management read/write operation timeout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_INVALID_STATE: PHY is in invalid state to perform requested operation</td>
</tr>
</tbody>
</table>

`phy_802_3_t *esp_eth_phy_into_phy_802_3 (esp_eth_phy_t *phy)`

Returns address to parent IEEE 802.3 PHY object infostructure.

<table>
<thead>
<tr>
<th>参数</th>
<th>phy</th>
<th>Ethernet PHY instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>返回</td>
<td>phy_802_3_t</td>
<td>address to parent IEEE 802.3 PHY object infostructure</td>
</tr>
</tbody>
</table>

*esp_err_t* `esp_eth_phy_802_3_obj_config_init (phy_802_3_t *phy_802_3, const eth_phy_config_t *config)`

Initializes configuration of parent IEEE 802.3 PHY object infostructure.

<table>
<thead>
<tr>
<th>参数</th>
<th>phy_802_3</th>
<th>Address to IEEE 802.3 PHY object infostructure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>config</td>
<td>Configuration of the IEEE 802.3 PHY object</td>
</tr>
<tr>
<td>返回</td>
<td></td>
<td>ESP_OK: configuration initialized successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESP_ERR_INVALID_ARG: invalid config argument</td>
</tr>
</tbody>
</table>
Chapter 2. API 参考

Structures

struct phy_802_3_t
IEEE 802.3 PHY object info structure.

Public Members

*esp_eth_phy_t parent
    Parent Ethernet PHY instance

*esp_eth_mediator_t *eth
    Mediator of Ethernet driver

int addr
    PHY address

uint32_t reset_timeout_ms
    Reset timeout value (Unit: ms)

uint32_t autonego_timeout_ms
    Auto-negotiation timeout value (Unit: ms)

eth_link_t link_status
    Current Link status

int reset_gpio_num
    Reset GPIO number, -1 means no hardware reset

Header File

  • components/esp_eth/include/esp_eth_netif_glue.h

Functions

esp_eth_netif_glue_handle_t esp_eth_new_netif_glue(esp_eth_handle_t eth_hdl)
Create a netif glue for Ethernet driver.

备注: netif glue is used to attach io driver to TCP/IP netif

参数 eth_hdl – Ethernet driver handle
返回 glue object, which inherits esp_netif_driver_base_t

esp_err_t esp_eth_del_netif_glue(esp_eth_netif_glue_handle_t eth_netif_glue)
Delete netif glue of Ethernet driver.

参数 eth_netif_glue – netif glue
返回 -ESP_OK: delete netif glue successfully

Type Definitions

typedef struct esp_eth_netif_glue_t *esp_eth_netif_glue_handle_t
Handle of netif glue - an intermediate layer between netif and Ethernet driver.
本部分的以太网 API 示例代码存放在 ESP-IDF 示例项目的 ethernet 目录下。
2.5.3 Thread

Introduction  Thread is an IP-based mesh networking protocol. It’s based on the 802.15.4 physical and MAC layer.

Application Examples  The openthread directory of ESP-IDF examples contains the following applications:

- The OpenThread interactive shell openthread/ot_cli.
- The Thread border router openthread/ot_br.
- The Thread radio co-processor openthread/ot_rcc.

API Reference  For manipulating the Thread network, the OpenThread api shall be used. The OpenThread api docs can be found at the OpenThread official website.

ESP-IDF provides extra apis for launching and managing the OpenThread stack, binding to network interfaces and border routing features.

Header File  
- components/openthread/include/esp_openthread.h

Functions

```c
esp_err_t esp_openthread_init(const esp_openthread_platform_config_t *init_config)
```

Initializes the full OpenThread stack.

备注: The OpenThread instance will also be initialized in this function.

参数  init_config  [in]  The initialization configuration.
返回
- ESP_OK on success
- ESP_ERR_NO_MEM if allocation has failed
- ESP_ERR_INVALID_ARG if radio or host connection mode not supported
- ESP_ERR_INVALID_STATE if already initialized

```c
esp_err_t esp_openthread_launch_mainloop(void)
```

Launches the OpenThread main loop.

备注: This function will not return unless error happens when running the OpenThread stack.

返回
- ESP_OK on success
- ESP_FAIL on other failures

```c
esp_err_t esp_openthread_deinit(void)
```

This function performs OpenThread stack and platform driver deinitialization.

返回
- ESP_OK on success
- ESP_ERR_INVALID_STATE if not initialized
otInstance *\texttt{esp\_openthread\_get\_instance} (void)

This function acquires the underlying OpenThread instance.

\textbf{备注：} This function can be called on other tasks without lock.

The OpenThread instance pointer

\textbf{Header File}

- components/openthread/include/esp_openthread_types.h

\textbf{Structures}

\textbf{struct} \texttt{esp\_openthread\_mainloop\_context\_t}

This structure represents a context for a select() based mainloop.

\textbf{Public Members}

\begin{itemize}
  \item \texttt{fd\_set read\_fds}
    \begin{itemize}
      \item The read file descriptors
    \end{itemize}
  \item \texttt{fd\_set write\_fds}
    \begin{itemize}
      \item The write file descriptors
    \end{itemize}
  \item \texttt{fd\_set error\_fds}
    \begin{itemize}
      \item The error file descriptors
    \end{itemize}
  \item \texttt{int max\_fd}
    \begin{itemize}
      \item The max file descriptor
    \end{itemize}
  \item \texttt{struct timeval timeout}
    \begin{itemize}
      \item The timeout
    \end{itemize}
\end{itemize}

\textbf{struct} \texttt{esp\_openthread\_uart\_config\_t}

The uart port config for OpenThread.

\textbf{Public Members}

\begin{itemize}
  \item \texttt{uart\_port\_t port}
    \begin{itemize}
      \item UART port number
    \end{itemize}
  \item \texttt{uart\_config\_t uart\_config}
    \begin{itemize}
      \item UART configuration, see \texttt{uart\_config\_t} docs
    \end{itemize}
  \item \texttt{int rx\_pin}
    \begin{itemize}
      \item UART RX pin
    \end{itemize}
\end{itemize}
int tx_pin
UART TX pin

struct esp_openthread_radio_config_t
The OpenThread radio configuration.

Public Members

datatype esp_openthread_radio_mode_t radio_mode
The radio mode

datatype esp_openthread_uart_config_t radio_uart_config
The uart configuration to RCP

struct esp_openthread_host_connection_config_t
The OpenThread host connection configuration.

Public Members

datatype esp_openthread_host_connection_mode_t host_connection_mode
The host connection mode

datatype esp_openthread_uart_config_t host_uart_config
The uart configuration to host

struct esp_openthread_port_config_t
The OpenThread port specific configuration.

Public Members

const char storage_partition_name
The partition for storing OpenThread dataset

uint8_t netif_queue_size
The packet queue size for the network interface

uint8_t task_queue_size
The task queue size

struct esp_openthread_platform_config_t
The OpenThread platform configuration.

Public Members

datatype esp_openthread_radio_config_t radio_config
The radio configuration
**esp_openthread_host_connection_config_t** host_config

The host connection configuration

**esp_openthread_port_config_t** port_config

The port configuration

### Type Definitions

typedef void (*esp_openthread_rcp_failure_handler)(void)

### Enumerations

enum **esp_openthread_event_t**

OpenThread event declarations.

**Values:**

- enumerator **OPENTHREAD_EVENT_START**
  OpenThread stack start
- enumerator **OPENTHREAD_EVENT_STOP**
  OpenThread stack stop
- enumerator **OPENTHREAD_EVENT_IF_UP**
  OpenThread network interface up
- enumerator **OPENTHREAD_EVENT_IF_DOWN**
  OpenThread network interface down
- enumerator **OPENTHREAD_EVENT_GOT_IP6**
  OpenThread stack added IPv6 address
- enumerator **OPENTHREAD_EVENT_LOST_IP6**
  OpenThread stack removed IPv6 address
- enumerator **OPENTHREAD_EVENT_MULTICAST_GROUP_JOIN**
  OpenThread stack joined IPv6 multicast group
- enumerator **OPENTHREAD_EVENT_MULTICAST_GROUP_LEAVE**
  OpenThread stack left IPv6 multicast group
- enumerator **OPENTHREAD_EVENT_TREL_ADD_IP6**
  OpenThread stack added TREL IPv6 address
- enumerator **OPENTHREAD_EVENT_TREL_REMOVE_IP6**
  OpenThread stack removed TREL IPv6 address
- enumerator **OPENTHREAD_EVENT_TREL_MULTICAST_GROUP_JOIN**
  OpenThread stack joined TREL IPv6 multicast group
enum \texttt{esp\_openthread\_radio\_mode\_t}

The radio mode of OpenThread.

\textit{Values:}

- \texttt{RADIO\_MODE\_NATIVE} \texttt{enumerator}
  
  Use the native 15.4 radio

- \texttt{RADIO\_MODE\_UART\_RCP} \texttt{enumerator}
  
  UART connection to a 15.4 capable radio co-processor (RCP)

- \texttt{RADIO\_MODE\_SPI\_RCP} \texttt{enumerator}
  
  SPI connection to a 15.4 capable radio co-processor (RCP)

enum \texttt{esp\_openthread\_host\_connection\_mode\_t}

How OpenThread connects to the host.

\textit{Values:}

- \texttt{HOST\_CONNECTION\_MODE\_NONE} \texttt{enumerator}
  
  Disable host connection

- \texttt{HOST\_CONNECTION\_MODE\_CLI\_UART} \texttt{enumerator}
  
  CLI UART connection to the host

- \texttt{HOST\_CONNECTION\_MODE\_RCP\_UART} \texttt{enumerator}
  
  RCP UART connection to the host

\textbf{Header File}

- components/openthread/include/esp_openthread_lock.h

\textbf{Functions}

\texttt{esp\_err\_t esp\_openthread\_lock\_init (void)}

This function initializes the OpenThread API lock.

\begin{itemize}
  \item ESP\_OK on success
  \item ESP\_ERR\_NO\_MEM if allocation has failed
  \item ESP\_ERR\_INVALID\_STATE if already initialized
\end{itemize}

\textbf{void esp\_openthread\_lock\_deinit (void)}

This function deinitializes the OpenThread API lock.

\textbf{bool esp\_openthread\_lock\_acquire (TickType\_t block\_ticks)}

This function acquires the OpenThread API lock.

\begin{itemize}
  \item ESP\_OK on success
  \item ESP\_ERR\_NO\_MEM if allocation has failed
  \item ESP\_ERR\_INVALID\_STATE if already initialized
\end{itemize}

\hrulewidth 0.5pt

\textbf{备注:} Every OT APIs that takes an otInstance argument MUST be protected with this API lock except that the call site is in OT callbacks.

\begin{itemize}
  \item block\_ticks \texttt{[in]} The maximum number of RTOS ticks to wait for the lock.
  \item True on lock acquired
\end{itemize}
Chapter 2. API

- False on failing to acquire the lock with the timeout.

```c
void esp_openthread_lock_release (void)
This function releases the OpenThread API lock.
```

**Header File**
- components/openthread/include/esp_openthread_netif_glue.h

**Functions**

```c
void* esp_openthread_netif_glue_init (const esp_openthread_platform_config_t *config)
This function initializes the OpenThread network interface glue.
```

参数 `config` [in] The platform configuration.
返回
- glue pointer on success
- NULL on failure

```c
void esp_openthread_netif_glue_deinit (void)
This function deinitializes the OpenThread network interface glue.
```

```c
esp_netif_t *esp_openthread_get_netif (void)
This function acquires the OpenThread netif.
```

返回 The OpenThread netif or NULL if not initialized.

**Macros**

```c
ESP_NETIF_INHERENT_DEFAULT_OPENTHREAD ()
Default configuration reference of OT esp-netif.
ESP_NETIF_DEFAULT_OPENTHREAD ()
```

**Header File**
- components/openthread/include/esp_openthread_border_router.h

**Functions**

```c
void esp_openthread_set_backbone_netif (esp_netif_t *backbone_netif)
Sets the backbone interface used for border routing.
```

备注: This function must be called before esp_openthread_init

参数 `backbone_netif` [in] The backbone network interface (WiFi or ethernet)

```c
esp_err_t esp_openthread_border_router_init (void)
Initializes the border router features of OpenThread.
```

备注: Calling this function will make the device behave as an OpenThread border router. Kconfig option CONFIG_OPENTHREAD_BORDER_ROUTER is required.

返回
- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if feature not supported
- ESP_ERR_INVALID_STATE if already initialized
- ESP_FAIL on other failures
**Chapter 2. API Reference**

`esp_err_t esp_openthread_border_router_deinit (void)`

Deinitializes the border router features of OpenThread.

- ESP_OK on success
- ESP_ERR_INVALID_STATE if not initialized
- ESP_FAIL on other failures

`esp_netif_t *esp_openthread_get_backbone_netif (void)`

Gets the backbone interface of OpenThread border router.

- The backbone interface or NULL if border router not initialized.

`void esp_openthread_register_rcp_failure_handler (esp_openthread_rcp_failure_handler handler)`

Registers the callback for RCP failure.

`void esp_openthread_rcp_deinit (void)`

Deinitializes the connection to RCP.

Thread 是一种基于 IPv6 的物联网网状网络技术。本部分的 Thread API 示例代码存放在 ESP-IDF 示例项目的 `openthread` 目录下。

### 2.5.4 IP 网络层协议

**ESP-NETIF**

The purpose of ESP-NETIF library is twofold:

- It provides an abstraction layer for the application on top of the TCP/IP stack. This will allow applications to choose between IP stacks in the future.
- The APIs it provides are thread safe, even if the underlying TCP/IP stack APIs are not.

ESP-IDF currently implements ESP-NETIF for the lwIP TCP/IP stack only. However, the adapter itself is TCP/IP implementation agnostic and different implementations are possible.

Some ESP-NETIF API functions are intended to be called by application code, for example to get/set interface IP addresses, configure DHCP. Other functions are intended for internal ESP-IDF use by the network driver layer.

In many cases, applications do not need to call ESP-NETIF APIs directly as they are called from the default network event handlers.

**ESP-NETIF architecture**

![ESP-NETIF Architecture Diagram]
Data and event flow in the diagram

- Initialization line from user code to ESP-NETIF and communication driver
- Data packets going from communication media to TCP/IP stack and back
- Events aggregated in ESP-NETIF propagates to driver, user code and network stack
- User settings and runtime configuration

ESP-NETIF interaction

A) User code, boiler plate

Overall application interaction with a specific IO driver for communication media and configured TCP/IP network stack is abstracted using ESP-NETIF APIs and outlined as below:

A) Initialization code

1) Initializes IO driver
2) Creates a new instance of ESP-NETIF and configure with

- ESP-NETIF specific options (flags, behaviour, name)
- Network stack options (netif init and input functions, not publicly available)
- IO driver specific options (transmit, free rx buffer functions, IO driver handle)
3) Attaches the IO driver handle to the ESP-NETIF instance created in the above steps

4) Configures event handlers
   • use default handlers for common interfaces defined in IO drivers; or define a specific handlers for
     customised behaviour/new interfaces
   • register handlers for app related events (such as IP lost/acquired)

B) Interaction with network interfaces using ESP-NETIF API
   • Getting and setting TCP/IP related parameters (DHCP, IP, etc)
   • Receiving IP events (connect/disconnect)
   • Controlling application lifecycle (set interface up/down)

B) Communication driver, IO driver, media driver
   Communication driver plays these two important roles in relation with ESP-NETIF:
   1) Event handlers: Define behaviour patterns of interaction with ESP-NETIF (for example: ethernet link-up -> turn netif on)
   2) Glue IO layer: Adapts the input/output functions to use ESP-NETIF transmit, receive and free receive buffer
      • Installs driver_transmit to appropriate ESP-NETIF object, so that outgoing packets from network stack are passed to the IO driver
      • Calls esp_netif_receive() to pass incoming data to network stack

C) ESP-NETIF
   ESP-NETIF is an intermediary between an IO driver and a network stack, connecting packet data path between these two. As that it provides a set of interfaces for attaching a driver to ESP-NETIF object (runtime) and configuring a network stack (compile time). In addition to that a set of API is provided to control network interface lifecycle and its TCP/IP properties. As an overview, the ESP-NETIF public interface could be divided into these 6 groups:
   1) Initialization APIs (to create and configure ESP-NETIF instance)
   2) Input/Output API (for passing data between IO driver and network stack)
   3) Event or Action API
      • Used for network interface lifecycle management
      • ESP-NETIF provides building blocks for designing event handlers
   4) Setters and Getters for basic network interface properties
   5) Network stack abstraction: enabling user interaction with TCP/IP stack
      • Set interface up or down
      • DHCP server and client API
      • DNS API
   6) Driver conversion utilities

D) Network stack
   Network stack has no public interaction with application code with regard to public interfaces and shall be fully abstracted by ESP-NETIF API.

E) ESP-NETIF L2 TAP Interface
   The ESP-NETIF L2 TAP interface is ESP-IDF mechanism utilized to access Data Link Layer (L2 per OSI/ISO) for frame reception and transmission from user application. Its typical usage in embedded world might be implementation of non-IP related protocols such as PTP, Wake on LAN and others. Note that only Ethernet (IEEE 802.3) is currently supported.
   From user perspective, the ESP-NETIF L2 TAP interface is accessed using file descriptors of VFS which provides a file-like interfacing (using functions like open(), read(), write(), etc). Refer to 虚拟文件系统组件 to learn more.
   There is only one ESP-NETIF L2 TAP interface device (pathname) available. However multiple file descriptors with different configuration can be opened at a time since the ESP-NETIF L2 TAP interface can be understood as generic entry point to Layer 2 infrastructure. Important is then specific configuration of particular file descriptor. It can be
configured to give an access to specific Network Interface identified by \texttt{if\_key} (e.g. \texttt{ETH\_DEF}) and to filter only specific frames based on their type (e.g. Ethernet type in case of IEEE 802.3). Filtering only specific frames is crucial since the ESP-NETIF L2 TAP needs to exist along with IP stack and so the IP related traffic (IP, ARP, etc.) should not be passed directly to the user application. Even though such option is still configurable, it is not recommended in standard use cases. Filtering is also advantageous from a perspective the user’s application gets access only to frame types it is interested in and the remaining traffic is either passed to other L2 TAP file descriptors or to IP stack.

**ESP-NETIF L2 TAP Interface Usage Manual**

**Initialization** To be able to use the ESP-NETIF L2 TAP interface, it needs to be enabled in Kconfig by \texttt{CONFIG_ESP_NETIF_L2_TAP} first and then registered by \texttt{esp\_vfs\_l2tap\_intf\_register()} prior usage of any VFS function.

**open()** Once the ESP-NETIF L2 TAP is registered, it can be opened at path name “/dev/net/tap”. The same path name can be opened multiple times up to \texttt{CONFIG_ESP_NETIF_L2_TAP_MAX_FDS} and multiple file descriptors with with different configuration may access the Data Link Layer frames.

The ESP-NETIF L2 TAP can be opened with \texttt{O\_NONBLOCK} file status flag to the \texttt{read()} does not block. Note that the \texttt{write()} may block in current implementation when accessing a Network interface since it is a shared resource among multiple ESP-NETIF L2 TAP file descriptors and IP stack, and there is currently no queuing mechanism deployed. The file status flag can be retrieved and modified using \texttt{fcntl()}.

On success, \texttt{open()} returns the new file descriptor (a nonnegative integer). On error, -1 is returned and \texttt{errno} is set to indicate the error.

**ioctl()** The newly opened ESP-NETIF L2 TAP file descriptor needs to be configured prior its usage since it is not bounded to any specific Network Interface and no frame type filter is configured. The following configuration options are available to do so:

- \texttt{L2TAP\_S\_INTF\_DEVICE} - bounds the file descriptor to specific Network Interface which is identified by its \texttt{if\_key}. ESP-NETIF Network Interface \texttt{if\_key} is passed to \texttt{ioctl()} as the third parameter. Note that default Network Interfaces \texttt{if\_key}’s used in ESP-IDF can be found in \texttt{esp\_netif\_include\_esp\_netif\_defaults.h}.
- \texttt{L2TAP\_S\_DEVICE\_DRV\_HNDL} - is other way how to bound the file descriptor to specific Network Interface. In this case the Network interface is identified directly by IO Driver handle (e.g. \texttt{esp\_eth\_handle\_t} in case of Ethernet). The IO Driver handle is passed to \texttt{ioctl()} as the third parameter.
- \texttt{L2TAP\_S\_RCV\_FILTER} - sets the filter to frames with this type to be passed to the file descriptor. In case of Ethernet frames, the frames are to be filtered based on Length/Ethernet type field. In case the filter value is set less than or equal to 0x05DC, the Ethernet type field is considered to represent IEEE802.3 Length Field and all frames with values in interval \textless 0, 0x05DC\textgreater at that field are to be passed to the file descriptor. The IEEE802.2 logical link control (LLC) resolution is then expected to be performed by user’s application. In case the filter value is set greater than 0x05DC, the Ethernet type field is considered to represent protocol identification and only frames which are equal to the set value are to be passed to the file descriptor.

All above set configuration options have getter counterpart option to read the current settings.

**Warning:** The file descriptor needs to be firstly bounded to specific Network Interface by \texttt{L2TAP\_S\_INTF\_DEVICE} or \texttt{L2TAP\_S\_DEVICE\_DRV\_HNDL} to be \texttt{L2TAP\_S\_RCV\_FILTER} option available.

**Note:** VLAN tagged frames are currently not recognized. If user needs to process VLAN tagged frames, they need set filter to be equal to VLAN tag (i.e. 0x8100 or 0x88A8) and process the VLAN tagged frames in user application.
Chapter 2. API 参考

备注：L2TAP_S_DEVICE_DRV_HNDL is particularly useful when user’s application does not require usage of IP stack and so ESP-NETIF is not required to be initialized too. As a result, Network Interface cannot be identified by its if_key and hence it needs to be identified directly by its IO Driver handle.

On success, ioctl() returns 0. On error, -1 is returned, and errno is set to indicate the error.

EBADF - not a valid file descriptor.

EACCES - option change is denied in this state (e.g. file descriptor has not been bounded to Network interface yet).

EINVAL - invalid configuration argument. Ethernet type filter is already used by other file descriptor on that same Network interface.

ENODEV - no such Network Interface which is tried to be assigned to the file descriptor exists.

ENOSYS - unsupported operation, passed configuration option does not exists.

fcntl() is used to manipulate with properties of opened ESP-NETIF L2 TAP file descriptor.

The following commands manipulate the status flags associated with file descriptor:

- F_GETFD - the function returns the file descriptor flags, the third argument is ignored.
- F_SETFD - sets the file descriptor flags to the value specified by the third argument. Zero is returned.

On error, -1 is returned, and errno is set to indicate the error.

EBADF - not a valid file descriptor.

ENOSYS - unsupported command.

read() Opened and configured ESP-NETIF L2 TAP file descriptor can be accessed by read() to get inbound frames. The read operation can be either blocking or non-blocking based on actual state of O_NONBLOCK file status flag. When the file status flag is set blocking, the read operation waits until a frame is received and context is switched to other task. When the file status flag is set non-blocking, the read operation returns immediately. In such case, either a frame is returned if it was already queued or the function indicates the queue is empty. The number of queued frames associated with one file descriptor is limited by CONFIG_ESP_NETIF_L2_TAP_RX_QUEUE_SIZE Kconfig option. Once the number of queued frames reach configured threshold, the newly arriving frames are dropped until the queue has enough room to accept incoming traffic (Tail Drop queue management).

On success, read() returns the number of bytes read. Zero is returned when size of the destination buffer is 0. On error, -1 is returned, and errno is set to indicate the error.

EBADF - not a valid file descriptor.

EAGAIN - the file descriptor has been marked non-blocking (O_NONBLOCK), and the read would block.

write() A raw Data Link Layer frame can be sent to Network Interface via opened and configured ESP-NETIF L2 TAP file descriptor. User’s application is responsible to construct the whole frame except for fields which are added automatically by the physical interface device. The following fields need to be constructed by the user’s application in case of Ethernet link: source/destination MAC addresses, Ethernet type, actual protocol header and user data. See below for more information about Ethernet frame structure.

```
+-------------------+-------------------+-------------+----------------------------
  | Destination MAC  | Source MAC        | Type/Length | Payload (protocol header/
  ← ---               ← ---                        data) ...                     
+-------------------+-------------------+-------------+----------------------------
  6B                 6B                  2B          0-1486B
```
In other words, there is no additional frame processing performed by the ESP-NETIF L2 TAP interface. It only checks the Ethernet type of the frame is the same as the filter configured in the file descriptor. If the Ethernet type is different, an error is returned and the frame is not sent. Note that the `write()` may block in current implementation when accessing a Network interface since it is a shared resource among multiple ESP-NETIF L2 TAP file descriptors and IP stack, and there is currently no queuing mechanism deployed.

On success, `write()` returns the number of bytes written. Zero is returned when size of the input buffer is 0. On error, -1 is returned, and `errno` is set to indicate the error.

**EBADF** - not a valid file descriptor.

**EBADMSG** - Ethernet type of the frame is different then file descriptor configured filter.

**EIO** - Network interface not available or busy.

### close()
Opened ESP-NETIF L2 TAP file descriptor can be closed by the `close()` to free its allocated resources. The ESP-NETIF L2 TAP implementation of `close()` may block. On the other hand, it is thread safe and can be called from different task than the file descriptor is actually used. If such situation occurs and one task is blocked in I/O operation and another task tries to close the file descriptor, the first task is unblocked. The first’s task read operation then ends with error.

On success, `close()` returns zero. On error, -1 is returned, and `errno` is set to indicate the error.

**EBADF** - not a valid file descriptor.

### select()
Select is used in a standard way, just `CONFIG_VFS_SUPPORT_SELECT` needs to be enabled to be the `select()` function available.

### ESP-NETIF programmer’s manual
Please refer to the example section for basic initialization of default interfaces:

- WiFi Station: `wifi/getting_started/station/main/station_example_main.c`
- Ethernet: `ethernet/basic/main/ethernet_example_main.c`
- L2 TAP: `protocols/l2tap/main/l2tap_main.c`
- WiFi Access Point: `wifi/getting_started/softAP/main/softap_example_main.c`

For more specific cases please consult this guide: `ESP-NETIF Custom I/O Driver`.

### WiFi default initialization
The initialization code as well as registering event handlers for default interfaces, such as softAP and station, are provided in separate APIs to facilitate simple startup code for most applications:

- `esp_netif_create_default_wifi_sta()`
- `esp_netif_create_default_wifi_ap()`

Please note that these functions return the `esp_netif` handle, i.e. a pointer to a network interface object allocated and configured with default settings, which as a consequence, means that:

- The created object has to be destroyed if a network de-initialization is provided by an application using `esp_netif_destroy_default_wifi()`.
- These default interfaces must not be created multiple times, unless the created handle is deleted using `esp_netif_destroy()`.
- When using Wifi in AP+STA mode, both these interfaces has to be created.

### API Reference
Chapter 2. API 参考

Header File

- components/esp_netif/include/esp_netif.h

Functions

`esp_err_t esp_netif_init (void)`
Initialize the underlying TCP/IP stack.

备注: This function should be called exactly once from application code, when the application starts up.

返回
- ESP_OK on success
- ESP_FAIL if initializing failed

`esp_err_t esp_netif_deinit (void)`
Deinitialize the esp-netif component (and the underlying TCP/IP stack)

Note: Deinitialization is not supported yet

返回
- ESP_ERR_INVALID_STATE if esp_netif not initialized
- ESP_ERR_NOT_SUPPORTED otherwise

`esp_netif_t *esp_netif_new (const esp_netif_config_t *esp_netif_config)`
Creates an instance of new esp-netif object based on provided config.

参数 esp_netif_config — pointer esp-netif configuration
返回
- pointer to esp-netif object on success
- NULL otherwise

`void esp_netif_destroy (esp_netif_t *esp_netif)`
Destroys the esp_netif object.

参数 esp_netif — [in] pointer to the object to be deleted

`esp_err_t esp_netif_set_driver_config (esp_netif_t *esp_netif, const esp_netif_driver_ifconfig_t *driver_config)`
Configures driver related options of esp_netif object.

参数
- esp_netif — [inout] pointer to the object to be configured
- driver_config — [in] pointer esp-netif io driver related configuration
返回
- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS if invalid parameters provided

`esp_err_t esp_netif_attach (esp_netif_t *esp_netif, esp_netif_iodriver_handle driver_handle)`
Attaches esp_netif instance to the io driver handle.

Calling this function enables connecting specific esp_netif object with already initialized io driver to update esp_netif object with driver specific configuration (i.e. calls post_attach callback, which typically sets io driver callbacks to esp_netif instance and starts the driver)

参数
- esp_netif — [inout] pointer to esp_netif object to be attached
- driver_handle — [in] pointer to the driver handle

返回
**Chapter 2. API Reference**

- ESP_OK on success
- ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED if driver’s pot_attach callback failed

```c
esp_err_t esp_netif_receive(esp_netif_t *esp_netif, void *buffer, size_t len, void *eb)
```

Passes the raw packets from communication media to the appropriate TCP/IP stack.

This function is called from the configured (peripheral) driver layer. The data are then forwarded as frames to the TCP/IP stack.

- **esp_netif** - [in] Handle to esp-netif instance
- **buffer** - [in] Received data
- **len** - [in] Length of the data frame
- **eb** - [in] Pointer to internal buffer (used in Wi-Fi driver)

**Return**
- ESP_OK

```c
void esp_netif_action_start(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)
```

Default building block for network interface action upon IO driver start event. Creates network interface, if AUTOUP enabled turns the interface on, if DHCPS enabled starts dhcp server.

**Remark:** This API can be directly used as event handler

```c
void esp_netif_action_stop(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)
```

Default building block for network interface action upon IO driver stop event.

**Remark:** This API can be directly used as event handler

```c
void esp_netif_action_connected(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)
```

Default building block for network interface action upon IO driver connected event.

**Remark:** This API can be directly used as event handler

```c
```
void **esp_netif_action_disconnected** (void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver disconnected event.

备注: This API can be directly used as event handler

参数
- esp_netif - [in] Handle to esp-netif instance
- base -
- event_id -
- data -

void **esp_netif_action_got_ip** (void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon network got IP event.

备注: This API can be directly used as event handler

参数
- esp_netif - [in] Handle to esp-netif instance
- base -
- event_id -
- data -

void **esp_netif_action_join_ip6_multicast_group** (void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 multicast group join.

备注: This API can be directly used as event handler

参数
- esp_netif - [in] Handle to esp-netif instance
- base -
- event_id -
- data -

void **esp_netif_action_leave_ip6_multicast_group** (void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 multicast group leave.

备注: This API can be directly used as event handler

参数
- esp_netif - [in] Handle to esp-netif instance
- base -
- event_id -
- data -

void **esp_netif_action_add_ip6_address** (void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 address added by the underlying stack.
Chapter 2. API 参考

备注：This API can be directly used as event handler

### 参数
- **esp_netif** - [in] Handle to esp-netif instance
- **base** -
- **event_id** -
- **data** -

void **esp_netif_action_remove_ip6_address** (void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IPv6 address removed by the underlying stack.

备注：This API can be directly used as event handler

### 参数
- **esp_netif** - [in] Handle to esp-netif instance
- **base** -
- **event_id** -
- **data** -

**esp_err_t esp_netif_set_default_netif** (**esp_netif_t** *esp_netif)

Manual configuration of the default netif.

This API overrides the automatic configuration of the default interface based on the route_prio. If the selected netif is set default using this API, no other interface could be set-default disregarding its route_prio number (unless the selected netif gets destroyed)

参数 **esp_netif** - [in] Handle to esp-netif instance
返回 ESP_OK on success

**esp_err_t esp_netif_set_mac** (**esp_netif_t** *esp_netif, uint8_t mac[])

Set the mac address for the interface instance.

参数 **esp_netif** - [in] Handle to esp-netif instance
参数 **mac** - [in] Desired mac address for the related network interface
返回
- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_NOT_SUPPORTED - mac not supported on this interface

**esp_err_t esp_netif_get_mac** (**esp_netif_t** *esp_netif, uint8_t mac[])

Get the mac address for the interface instance.

参数 **esp_netif** - [in] Handle to esp-netif instance
参数 **mac** - [out] Resultant mac address for the related network interface
返回
- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_NOT_SUPPORTED - mac not supported on this interface

**esp_err_t esp_netif_set_hostname** (**esp_netif_t** *esp_netif, const char *hostname)

Set the hostname of an interface.

The configured hostname overrides the default configuration value CONFIG_LWIP_LOCAL_HOSTNAME. Please note that when the hostname is altered after interface started/connected the changes would only be reflected once the interface restarts/reconnects.
Chapter 2. API 参考

参数
- esp_netif – [in] Handle to esp-netif instance

返回
- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_ESP_NETIF_INVALID_PARAMS - parameter error

**esp_err_t** esp_netif_get_hostname (esp_netif_t *esp_netif, const char **hostname)

Get interface hostname.

参数
- esp_netif – [in] Handle to esp-netif instance
- hostname – [out] Returns a pointer to the hostname. May be NULL if no hostname is set. If set non-NULL, pointer remains valid (and string may change if the hostname changes).

返回
- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_ESP_NETIF_INVALID_PARAMS - parameter error

**bool** esp_netif_is_netif_up (esp_netif_t *esp_netif)

Test if supplied interface is up or down.

参数 esp_netif – [in] Handle to esp-netif instance

返回
- true - Interface is up
- false - Interface is down

**esp_err_t** esp_netif_get_ip_info (esp_netif_t *esp_netif, esp_netif_ip_info_t *ip_info)

Get interface’s IP address information.

If the interface is up, IP information is read directly from the TCP/IP stack. If the interface is down, IP information is read from a copy kept in the ESP-NETIF instance

参数
- esp_netif – [in] Handle to esp-netif instance
- ip_info – [out] If successful, IP information will be returned in this argument.

返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

**esp_err_t** esp_netif_get_old_ip_info (esp_netif_t *esp_netif, esp_netif_ip_info_t *ip_info)

Get interface’s old IP information.

Returns an “old” IP address previously stored for the interface when the valid IP changed.

If the IP lost timer has expired (meaning the interface was down for longer than the configured interval) then the old IP information will be zero.

参数
- esp_netif – [in] Handle to esp-netif instance
- ip_info – [out] If successful, IP information will be returned in this argument.

返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

**esp_err_t** esp_netif_set_ip_info (esp_netif_t *esp_netif, const esp_netif_ip_info_t *ip_info)

Set interface’s IP address information.

This function is mainly used to set a static IP on an interface.

If the interface is up, the new IP information is set directly in the TCP/IP stack.
The copy of IP information kept in the ESP-NETIF instance is also updated (this copy is returned if the IP is queried while the interface is still down.)

备注: DHCP client/server must be stopped (if enabled for this interface) before setting new IP information.

备注: Calling this interface for may generate a SYSTEM_EVENT_STA_GOT_IP or SYSTEM_EVENT_ETH_GOT_IP event.

参数
- esp_netif --[in] Handle to esp-netif instance
- ip_info --[in] IP information to set on the specified interface

返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED If DHCP server or client is still running

\[
\text{esp_err_t } \text{esp_netif_set_old_ip_info}(\text{esp_netif_t *esp_netif, const esp_netif_ip_info_t *ip_info})
\]

Set interface old IP information.

This function is called from the DHCP client (if enabled), before a new IP is set. It is also called from the default handlers for the SYSTEM_EVENT_STA_CONNECTED and SYSTEM_EVENT_ETH_CONNECTED events.

Calling this function stores the previously configured IP, which can be used to determine if the IP changes in the future.

If the interface is disconnected or down for too long, the “IP lost timer” will expire (after the configured interval) and set the old IP information to zero.

参数
- esp_netif --[in] Handle to esp-netif instance
- ip_info --[in] Store the old IP information for the specified interface

返回
- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

\[
\text{int } \text{esp_netif_get_netif_impl_index}(\text{esp_netif_t *esp_netif})
\]

Get net interface index from network stack implementation.

备注: This index could be used in setsockopt() to bind socket with multicast interface

参数 esp_netif --[in] Handle to esp-netif instance

返回 implementation specific index of interface represented with supplied esp_netif

\[
\text{esp_err_t } \text{esp_netif_get_netif_impl_name}(\text{esp_netif_t *esp_netif, char *name})
\]

Get net interface name from network stack implementation.

备注: This name could be used in setsockopt() to bind socket with appropriate interface

参数
- esp_netif --[in] Handle to esp-netif instance
Chapter 2. API

- **name** [out] Interface name as specified in underlying TCP/IP stack. Note that the actual name will be copied to the specified buffer, which must be allocated to hold maximum interface name size (6 characters for lwIP)

**esp_err_t esp_netif_dhcps_option (esp_netif_t *esp_netif, esp_netif_dhcp_option_mode_t opt_op, esp_netif_dhcp_option_id_t opt_id, void *opt_val, uint32_t opt_len)**

Set or Get DHCP server option.

**参数**

- **esp_netif** [in] Handle to esp-netif instance
- **opt_op** [in] ESP_NETIF_OP_SET to set an option, ESP_NETIF_OP_GET to get an option.
- **opt_id** [in] Option index to get or set, must be one of the supported enum values.
- **opt_val** [inout] Pointer to the option parameter.
- **opt_len** [in] Length of the option parameter.

**返回**

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

**esp_err_t esp_netif_dhcpc_option (esp_netif_t *esp_netif, esp_netif_dhcp_option_mode_t opt_op, esp_netif_dhcp_option_id_t opt_id, void *opt_val, uint32_t opt_len)**

Set or Get DHCP client option.

**参数**

- **esp_netif** [in] Handle to esp-netif instance
- **opt_op** [in] ESP_NETIF_OP_SET to set an option, ESP_NETIF_OP_GET to get an option.
- **opt_id** [in] Option index to get or set, must be one of the supported enum values.
- **opt_val** [inout] Pointer to the option parameter.
- **opt_len** [in] Length of the option parameter.

**返回**

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

**esp_err_t esp_netif_dhcpc_start (esp_netif_t *esp_netif)**

Start DHCP client (only if enabled in interface object)

**esp_err_t esp_netif_dhcpc_stop (esp_netif_t *esp_netif)**

Stop DHCP client (only if enabled in interface object)
备注：Calling action_netif_stop() will also stop the DHCP Client if it is running.

参数 esp_netif –[in] Handle to esp-netif instance
返回
• ESP_OK
• ESP_ERR_ESP_NETIF_INVALID_PARAMS
• ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
• ESP_ERR_ESP_NETIF_IF_NOT_READY

esp_err_t esp_netif_dhcpc_get_status (esp_netif_t *esp_netif, esp_netif_dhcp_status_t *status)
Get DHCP client status.

参数
• esp_netif –[in] Handle to esp-netif instance
• status –[out] If successful, the status of DHCP client will be returned in this argument.
返回
• ESP_OK

esp_err_t esp_netif_dhcps_get_status (esp_netif_t *esp_netif, esp_netif_dhcp_status_t *status)
Get DHCP Server status.

参数
• esp_netif –[in] Handle to esp-netif instance
• status –[out] If successful, the status of the DHCP server will be returned in this argument.
返回
• ESP_OK

esp_err_t esp_netif_dhcps_start (esp_netif_t *esp_netif)
Start DHCP server (only if enabled in interface object)

参数 esp_netif –[in] Handle to esp-netif instance
返回
• ESP_OK
• ESP_ERR_ESP_NETIF_INVALID_PARAMS
• ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

esp_err_t esp_netif_dhcps_stop (esp_netif_t *esp_netif)
Stop DHCP server (only if enabled in interface object)

参数 esp_netif –[in] Handle to esp-netif instance
返回
• ESP_OK
• ESP_ERR_ESP_NETIF_INVALID_PARAMS
• ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
• ESP_ERR_ESP_NETIF_IF_NOT_READY

esp_err_t esp_netif_dhcps_get_clients_by_mac (esp_netif_t *esp_netif, int num,
    esp_netif_pair_mac_ip_t *mac_ip_pair)
Populate IP addresses of clients connected to DHCP server listed by their MAC addresses.

参数
• esp_netif –[in] Handle to esp-netif instance
• num –[in] Number of clients with specified MAC addresses in the array of pairs
• mac_ip_pair –[inout] Array of pairs of MAC and IP addresses (MAC are inputs, IP outputs)
返回
• ESP_OK on success
• ESP_ERR_ESP_NETIF_INVALID_PARAMS on invalid params
• ESP_ERR_NOT_SUPPORTED if DHCP server not enabled

```c
esp_err_t esp_netif_set_dns_info(esp_netif_t *esp_netif, esp_netif_dns_type_t type, esp_netif_dns_info_t *dns)
```

Set DNS Server information.

This function behaves differently if DHCP server or client is enabled.

If DHCP client is enabled, main and backup DNS servers will be updated automatically from the DHCP lease if the relevant DHCP options are set. Fallback DNS Server is never updated from the DHCP lease and is designed to be set via this API. If DHCP client is disabled, all DNS server types can be set via this API only.

If DHCP server is enabled, the Main DNS Server setting is used by the DHCP server to provide a DNS Server option to DHCP clients (Wi-Fi stations).

- The default Main DNS server is typically the IP of the DHCP server itself.
- This function can override it by setting server type ESP_NETIF_DNS_MAIN.
- Other DNS Server types are not supported for the DHCP server.
- To propagate the DNS info to client, please stop the DHCP server before using this API.

### 参数

- **esp_netif** [in] Handle to esp-netif instance
- **type** [in] Type of DNS Server to set: ESP_NETIF_DNS_MAIN, ESP_NETIF_DNS_BACKUP, ESP_NETIF_DNS_FALLBACK
- **dns** [in] DNS Server address to set

### 返回

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS invalid params

```c
esp_err_t esp_netif_get_dns_info(esp_netif_t *esp_netif, esp_netif_dns_type_t type, esp_netif_dns_info_t *dns)
```

Get DNS Server information.

Return the currently configured DNS Server address for the specified interface and Server type.

This may be result of a previous call to `esp_netif_set_dns_info()`. If the interface’s DHCP client is enabled, the Main or Backup DNS Server may be set by the current DHCP lease.

### 参数

- **esp_netif** [in] Handle to esp-netif instance
- **type** [in] Type of DNS Server to get: ESP_NETIF_DNS_MAIN, ESP_NETIF_DNS_BACKUP, ESP_NETIF_DNS_FALLBACK
- **dns** [out] DNS Server result is written here on success

### 返回

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS invalid params

```c
esp_err_t esp_netif_create_ip6_linklocal(esp_netif_t *esp_netif)
```

Create interface link-local IPv6 address.

Cause the TCP/IP stack to create a link-local IPv6 address for the specified interface.

This function also registers a callback for the specified interface, so that if the link-local address becomes verified as the preferred address then a SYSTEM_EVENT_GOT_IP6 event will be sent.

### 参数 **esp_netif** [in] Handle to esp-netif instance

### 返回

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

```c
esp_err_t esp_netif_get_ip6_linklocal(esp_netif_t *esp_netif, esp_ip6_addr_t *if_ip6)
```

Get interface link-local IPv6 address.
If the specified interface is up and a preferred link-local IPv6 address has been created for the interface, return a copy of it.

参数
- esp_netif –[in] Handle to esp-netif instance
- if_ip6 –[out] IPv6 information will be returned in this argument if successful.

返回
- ESP_OK
- ESP_FAIL If interface is down, does not have a link-local IPv6 address, or the link-local IPv6 address is not a preferred address.

esp_err_t esp_netif_get_ip6_global (esp_netif_t *esp_netif, esp_ip6_addr_t *if_ip6)
Get interface global IPv6 address.

If the specified interface is up and a preferred global IPv6 address has been created for the interface, return a copy of it.

参数
- esp_netif –[in] Handle to esp-netif instance
- if_ip6 –[out] IPv6 information will be returned in this argument if successful.

返回
- ESP_OK
- ESP_FAIL If interface is down, does not have a global IPv6 address, or the global IPv6 address is not a preferred address.

int esp_netif_get_all_ip6 (esp_netif_t *esp_netif, esp_ip6_addr_t if_ip6[])
Get all IPv6 addresses of the specified interface.

参数
- esp_netif –[in] Handle to esp-netif instance
- if_ip6 –[out] Array of IPv6 addresses will be copied to the argument

返回 number of returned IPv6 addresses

void esp_netif_set_ip4_addr (esp_ip4_addr_t *addr, uint8_t a, uint8_t b, uint8_t c, uint8_t d)
Sets IPv4 address to the specified octets.

参数
- addr –[out] IP address to be set
- a –the first octet (127 for IP 127.0.0.1)
- b –
- c –
- d –

char *esp_ip4addr_ntoa (const esp_ip4_addr_t *addr, char *buf, int buflen)
Converts numeric IP address into decimal dotted ASCII representation.

参数
- addr –ip address in network order to convert
- buf –target buffer where the string is stored
- buflen –length of buf

返回 either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

uint32_t esp_ip4addr_aton (const char *addr)
Ascii internet address interpretation routine The value returned is in network order.

参数 addr –IP address in ascii representation (e.g. “127.0.0.1” )
返回 ip address in network order

esp_err_t esp_netif_str_to_ip4 (const char *src, esp_ip4_addr_t *dst)
Converts Ascii internet IPv4 address into esp_ip4_addr_t.

参数
- src –[in] IPv4 address in ascii representation (e.g. “127.0.0.1” )
Chapter 2. API

Esperrf Systems 942 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback

• dst –[out] Address of the target esp_ip4_addr_t structure to receive converted address

ESP_OK on success
ESP_FAIL if conversion failed
ESP_ERR_INVALID_ARG if invalid parameter is passed into

esp_err_t esp_netif_str_to_ip6 (const char *src, esp_ip6_addr_t *dst)
Converts Ascii internet IPv6 address into esp_ip4_addr_t. Zeros in the IP address can be stripped or completely omitted: “2001:db8:85a3:0:0:0:2:1” or “2001:db8::2:1”

src –[in] IPv6 address in ascii representation (e.g. “2001:0db8:85a3:0000:0000:0000:0002:0001”)

dst –[out] Address of the target esp_ip6_addr_t structure to receive converted address

ESP_OK on success
ESP_FAIL if conversion failed
ESP_ERR_INVALID_ARG if invalid parameter is passed into

esp_netif_iodriver_handle esp_netif_get_io_driver (esp_netif_t *esp_netif)
Gets media driver handle for this esp-netif instance.

esp_netif_t *esp_netif_get_handle_from_ifkey (const char *if_key)
Searches over a list of created objects to find an instance with supplied if key.

if_key –Textual description of network interface

don –Handle to esp-netif instance

esp_netif_flags_t esp_netif_get_flags (esp_netif_t *esp_netif)
Returns configured flags for this interface.

esp_netif_t –[in] Handle to esp-netif instance

Configuration flags

const char *esp_netif_get_ifkey (esp_netif_t *esp_netif)
Returns configured interface key for this esp-netif instance.

esp_netif_t –[in] Handle to esp-netif instance

Textual description of related interface

const char *esp_netif_get_desc (esp_netif_t *esp_netif)
Returns configured interface type for this esp-netif instance.

esp_netif_t –[in] Handle to esp-netif instance

Enumerated type of this interface, such as station, AP, ethernet

int esp_netif_get_route_prio (esp_netif_t *esp_netif)
Returns configured routing priority number.

esp_netif_t –[in] Handle to esp-netif instance

Integer representing the instance’s route-prio, or -1 if invalid parameters

int32_t esp_netif_get_event_id (esp_netif_t *esp_netif, esp_netif_ip_event_type_t event_type)
Returns configured event for this esp-netif instance and supplied event type.

esp_netif_t –[in] Handle to esp-netif instance

event_type – (either get or lost IP)

specific event id which is configured to be raised if the interface lost or acquired IP address

-1 if supplied event_type is not known
**esp_netif_t** *esp_netif_next (esp_netif_t *esp_netif)

Iterates over list of interfaces. Returns first netif if NULL given as parameter.

- **参数** esp_netif - [in] Handle to esp-netif instance
- **返回** First netif from the list if supplied parameter is NULL, next one otherwise

**size_t esp_netif_get_nr_of_ifs (void)**

Returns number of registered esp_netif objects.

- **返回** Number of esp_netifs

**void esp_netif_netstack_buf_ref (void *netstack_buf)**

Increases the reference counter of net stack buffer

- **参数** netstack_buf - [in] the net stack buffer

**void esp_netif_netstack_buf_free (void *netstack_buf)**

Frees the netstack buffer

- **参数** netstack_buf - [in] the net stack buffer

**Header File**

- components/esp_netif/include/esp_netif_types.h

**Structures**

**struct esp_netif_dns_info_t**

DNS server info.

**Public Members**

**esp_ip_addr_t ip**

IPV4 address of DNS server

**struct esp_netif_ip_info_t**

Event structure for IP_EVENT_STA_GOT_IP, IP_EVENT_ETH_GOT_IP events

**Public Members**

**esp_ip4_addr_t ip**

Interface IPV4 address

**esp_ip4_addr_t netmask**

Interface IPV4 netmask

**esp_ip4_addr_t gw**

Interface IPV4 gateway address

**struct esp_netif_ip6_info_t**

IPV6 IP address information.
Public Members

*esp_ip6_addr_t* `ip`
- Interface IPV6 address

struct `ip_event_got_ip_t`
- Event structure for IP_EVENT_GOT_IP event.

Public Members

*esp_netif_t* `*esp_netif`
- Pointer to corresponding esp-netif object

*esp_netif_ip_info_t* `ip_info`
- IP address, netmask, gateway IP address

bool `ip_changed`
- Whether the assigned IP has changed or not

struct `ip_event_got_ip6_t`
- Event structure for IP_EVENT_GOT_IP6 event

Public Members

*esp_netif_t* `*esp_netif`
- Pointer to corresponding esp-netif object

*esp_netif_ip6_info_t* `ip6_info`
- IPv6 address of the interface

int `ip_index`
- IPv6 address index

struct `ip_event_add_ip6_t`
- Event structure for ADD_IP6 event

Public Members

*esp_ip6_addr_t* `addr`
- The address to be added to the interface

bool `preferred`
- The default preference of the address

struct `ip_event_ap_staipassigned_t`
- Event structure for IP_EVENT_AP_STAIPASSIGNED event
Chapter 2. API 参考

Public Members

\textit{esp_netif_t *} \textit{esp_netif}

  Pointer to the associated netif handle

\textit{esp_ip4_addr_t} \textit{ip}

  IP address which was assigned to the station

uint8_t \textit{mac}[6]

  MAC address of the connected client

struct \textit{bridgeif_config}

  LwIP bridge configuration

Public Members

uint16_t \textit{max_fdb_dyn_entries}

  maximum number of entries in dynamic forwarding database

uint16_t \textit{max_fdb_sta_entries}

  maximum number of entries in static forwarding database

uint8_t \textit{max_ports}

  maximum number of ports the bridge can consist of

struct \textit{esp_netif_inherent_config}

  ESP-netif inherent config parameters.

Public Members

\textit{esp_netif_flags_t} \textit{flags}

  flags that define esp-netif behavior

uint8_t \textit{mac}[6]

  initial mac address for this interface

const \textit{esp_netif_ip_info_t *} \textit{ip_info}

  initial ip address for this interface

uint32_t \textit{get_ip_event}

  event id to be raised when interface gets an IP

uint32_t \textit{lost_ip_event}

  event id to be raised when interface losts its IP

const char \textit{*if_key}

  string identifier of the interface
const char *if_desc
    textual description of the interface

int route_prio
    numeric priority of this interface to become a default routing if (if other netifs are up). A higher value of route_prio indicates a higher priority

bridgeif_config_t *bridge_info
    LwIP bridge configuration

struct esp_netif_driver_base_s
    ESP-netif driver base handle.

Public Members

esp_err_t (*post_attach)(esp_netif_t *netif, esp_netif_iodriver_handle h)
    post attach function pointer

esp_netif_t *netif
    netif handle

struct esp_netif_driver_ifconfig
    Specific IO driver configuration.

Public Members

esp_netif_iodriver_handle handle
    io-driver handle

esp_err_t (*transmit)(void *h, void *buffer, size_t len)
    transmit function pointer

esp_err_t (*transmit_wrap)(void *h, void *buffer, size_t len, void *netstack_buffer)
    transmit wrap function pointer

void (*driver_free_rx_buffer)(void *h, void *buffer)
    free rx buffer function pointer

struct esp_netif_config
    Generic esp_netif configuration.

Public Members

const esp_netif_inherent_config_t *base
    base config
const esp_netif_driver_ifconfig_t *driver
driver config

const esp_netif_netstack_config_t *stack
stack config

struct esp_netif_pair_mac_ip_t
DHCP client’s addr info (pair of MAC and IP address)

Public Members

uint8_t mac[6]
Clients MAC address

esp_ip4_addr_t ip
Clients IP address

Macros

ESP_ERR_ESP_NETIF_BASE
Definition of ESP-NETIF based errors.

ESP_ERR_ESP_NETIF_INVALID_PARAMS

ESP_ERR_ESP_NETIF_IF_NOT_READY

ESP_ERR_ESP_NETIF_DHCPC_START_FAILED

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED

ESP_ERR_ESP_NETIF_NO_MEM

ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED

ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED

ESP_ERR_ESP_NETIF_INIT_FAILED

ESP_ERR_ESP_NETIF_DNS_NOT_CONFIGURED

ESP_ERR_ESP_NETIF_MLD6_FAILED

ESP_ERR_ESP_NETIF_IP6_ADDR_FAILED
**Chapter 2. API Reference**

- **ESP_ERR_ESP_NETIF_DHCPS_START_FAILED**
  - Definition of ESP-NETIF bridge control.

- **ESP_NETIF_BR_FLOOD**

- **ESP_NETIF_BR_DROP**

- **ESP_NETIF_BR_FDW_CPU**

**Type Definitions**

typedef struct esp_netif_obj esp_netif_t

typedef enum esp_netif_flags esp_netif_flags_t

typedef enum esp_netif_ip_event_type esp_netif_ip_event_type_t

typedef struct bridgeif_config bridgeif_config_t
  LwIP bridge configuration

typedef struct esp_netif_inherent_config esp_netif_inherent_config_t
  ESP-netif inherent config parameters.

typedef struct esp_netif_config esp_netif_config_t

typedef void *esp_netif_iodriver_handle
  IO driver handle type.

typedef struct esp_netif_driver_base_s esp_netif_driver_base_t
  ESP-netif driver base handle.

typedef struct esp_netif_driver_ifconfig esp_netif_driver_ifconfig_t

typedef struct esp_netif_netstack_config esp_netif_netstack_config_t
  Specific L3 network stack configuration.

typedef esp_err_t (*esp_netif_receive_t)(esp_netif_t *esp_netif, void *buffer, size_t len, void *eb)
  ESP-NETIF Receive function type.

**Enumerations**

enum esp_netif_dns_type_t
  Type of DNS server.

  Values:

  - enumerator ESP_NETIF_DNS_MAIN
    DNS main server address

---

Submit Document Feedback
enumerator **ESP_NETIF_DNS_BACKUP**
  DNS backup server address (Wi-Fi STA and Ethernet only)

enumerator **ESP_NETIF_DNS_FALLBACK**
  DNS fallback server address (Wi-Fi STA and Ethernet only)

enumerator **ESP_NETIF_DNS_MAX**

enum **esp_netif_dhcp_status_t**
  Status of DHCP client or DHCP server.
  **Values:**

  enumerator **ESP_NETIF_DHCP_INIT**
  DHCP client/server is in initial state (not yet started)

  enumerator **ESP_NETIF_DHCP_STARTED**
  DHCP client/server has been started

  enumerator **ESP_NETIF_DHCP_STOPPED**
  DHCP client/server has been stopped

  enumerator **ESP_NETIF_DHCP_STATUS_MAX**

enum **esp_netif_dhcp_option_mode_t**
  Mode for DHCP client or DHCP server option functions.
  **Values:**

  enumerator **ESP_NETIF_OP_START**

  enumerator **ESP_NETIF_OP_SET**
  Set option

  enumerator **ESP_NETIF_OP_GET**
  Get option

  enumerator **ESP_NETIF_OP_MAX**

enum **esp_netif_dhcp_option_id_t**
  Supported options for DHCP client or DHCP server.
  **Values:**

  enumerator **ESP_NETIF_SUBNET_MASK**
  Network mask

  enumerator **ESP_NETIF_DOMAIN_NAME_SERVER**
  Domain name server
enumerator **ESP_NETIF_ROUTER_SOLICITATION_ADDRESS**

Solicitation router address

enumerator **ESP_NETIF_REQUESTED_IP_ADDRESS**

Request specific IP address

enumerator **ESP_NETIF_IP_ADDRESSLEASE_TIME**

Request IP address lease time

enumerator **ESP_NETIF_IP_REQUEST_RETRY_TIME**

Request IP address retry counter

enumerator **ESP_NETIF_VENDOR_CLASS_IDENTIFIER**

Vendor Class Identifier of a DHCP client

enumerator **ESP_NETIF_VENDOR_SPECIFIC_INFO**

Vendor Specific Information of a DHCP server

enum **ip_event_t**

IP event declarations

*Values:*

enumerator **IP_EVENT_STA_GOT_IP**

station got IP from connected AP

enumerator **IP_EVENT_STA_LOST_IP**

station lost IP and the IP is reset to 0

enumerator **IP_EVENT_AP_STAIPASSIGNED**

soft-AP assign an IP to a connected station

enumerator **IP_EVENT_GOT_IP6**

station or ap or ethernet interface v6IP addr is preferred

enumerator **IP_EVENT_ETH_GOT_IP**

ethernet got IP from connected AP

enumerator **IP_EVENT_ETH_LOST_IP**

ethernet lost IP and the IP is reset to 0

enumerator **IP_EVENT_PPP_GOT_IP**

PPP interface got IP

enumerator **IP_EVENT_PPP_LOST_IP**

PPP interface lost IP

enum **esp_netif_flags**

*Values:*
Chapter 2. API

enumerator ESP_NETIF_DHCP_CLIENT
enumerator ESP_NETIF_DHCP_SERVER
enumerator ESP_NETIF_FLAG_AUTOUP
enumerator ESP_NETIF_FLAG_GARP
enumerator ESP_NETIF_FLAG_EVENT_IP_MODIFIED
enumerator ESP_NETIF_FLAG_IS_PPP
enumerator ESP_NETIF_FLAG_IS_BRIDGE

enum esp_netif_ip_event_type
Values:
enumerator ESP_NETIF_IP_EVENT_GOT_IP
enumerator ESP_NETIF_IP_EVENT_LOST_IP

Header File
• components/esp_netif/include/esp_netif_ip_addr.h

Functions

esp_ip6_addr_type_t esp_netif_ip6_get_addr_type(espl_ip6_addr_t *ip6_addr)
Get the IPv6 address type.

static inline void esp_netif_ip_addr_copy(espl_ip_addr_t *dest, const espl_ip_addr_t *src)
Copy IP addresses.

Parameters
• dest - [out] destination IP
• src - [in] source IP

Structures

struct esp_ip6_addr
IPv6 address.

Public Members

uint32_t addr[4]
IPv6 address
uint8_t zone  
zone ID

struct esp_ip4_addr  
IPv4 address.

**Public Members**

uint32_t addr  
IPv4 address

struct _ip_addr  
IP address.

**Public Members**

*esp_ip6_addr_t* ip6  
IPv6 address type

*esp_ip4_addr_t* ip4  
IPv4 address type

union _ip_addr::{anonymous} u_addr  
IP address union

uint8_t type  
ipaddress type

**Macros**

`esp_netif_htonl` (x)

`esp_netif_ip4_makeu32` (a, b, c, d)

`ESP_IP6_ADDR_BLOCK1` (ip6addr)

`ESP_IP6_ADDR_BLOCK2` (ip6addr)

`ESP_IP6_ADDR_BLOCK3` (ip6addr)

`ESP_IP6_ADDR_BLOCK4` (ip6addr)

`ESP_IP6_ADDR_BLOCK5` (ip6addr)

`ESP_IP6_ADDR_BLOCK6` (ip6addr)

`ESP_IP6_ADDR_BLOCK7` (ip6addr)

`ESP_IP6_ADDR_BLOCK8` (ip6addr)

**IPSTR**

`esp_ip4_addr_get_byte` (ipaddr, idx)
Chapter 2. API

```c
esp_ip4_addr (ipaddr)
esp_ip4_addr2 (ipaddr)
esp_ip4_addr3 (ipaddr)
esp_ip4_addr4 (ipaddr)
esp_ip4_addr1_16 (ipaddr)
esp_ip4_addr2_16 (ipaddr)
esp_ip4_addr3_16 (ipaddr)
esp_ip4_addr4_16 (ipaddr)
IP2STR (ipaddr)
IPV6STR
IPV62STR (ipaddr)

ESP_IPADDR_TYPE_V4

ESP_IPADDR_TYPE_V6

ESP_IPADDR_TYPE_ANY

ESP_IP4TOUINT32 (a, b, c, d)
ESP_IP4TOADDR (a, b, c, d)
ESP_IP4ADDR_INIT (a, b, c, d)
ESP_IP6ADDR_INIT (a, b, c, d)

Type Definitions

typedef struct esp_ip4_addr esp_ip4_addr_t

typedef struct esp_ip6_addr esp_ip6_addr_t

typedef struct _ip_addr esp_ip_addr_t
    IP address.

Enumerations

enum esp_ip6_addr_type_t
    Values:

    enumerator ESP_IP6_ADDR_IS_UNKNOWN

    enumerator ESP_IP6_ADDR_IS_GLOBAL

    enumerator ESP_IP6_ADDR_IS_LINK_LOCAL
```
enumerator ESP_IP6_ADDR_IS_SITE_LOCAL
enumerator ESP_IP6_ADDR_IS_UNIQUE_LOCAL
enumerator ESP_IP6_ADDR_IS_IPV4_MAPPED_IPV6

Header File
• components/esp_netif/include/esp_vfs_l2tap.h

Functions
`esp_err_t esp_vfs_l2tap_intf_register (l2tap_vfs_config_t *config)`
Add L2 TAP virtual filesystem driver.
This function must be called prior usage of ESP-NETIF L2 TAP Interface
参数 config - L2 TAP virtual filesystem driver configuration. Default base path /dev/net/tap is used when this parameter is NULL.
返回 esp_err_t
• ESP_OK on success

`esp_err_t esp_vfs_l2tap_intf_unregister (const char*base_path)`
Removes L2 TAP virtual filesystem driver.
参数 base_path - Base path to the L2 TAP virtual filesystem driver. Default path /dev/net/tap is used when this parameter is NULL.
返回 esp_err_t
• ESP_OK on success

`esp_err_t esp_vfs_l2tap_eth_filter (l2tap_iodriver_handle driver_handle, void *buff, size_t *size)`
Filters received Ethernet L2 frames into L2 TAP infrastructure.
参数
• driver_handle - handle of driver at which the frame was received
• buff - received L2 frame
• size - input length of the L2 frame which is set to 0 when frame is filtered into L2 TAP
返回 esp_err_t
• ESP_OK is always returned

Structures
`struct l2tap_vfs_config_t`
L2Tap VFS config parameters.

Public Members

const char *base_path
vfs base path

Macros

L2TAP_VFS_DEFAULT_PATH
L2TAP_VFS_CONFIG_DEFAULT ()
**Type Definitions**

typedef void *l2tap_iodriver_handle

**Enumerations**

eenum l2tap_ioctl_opt_t
    Values:
    
    enumerator L2TAP_S_RCV_FILTER
    enumerator L2TAP_G_RCV_FILTER
    enumerator L2TAP_S_INTF_DEVICE
    enumerator L2TAP_G_INTF_DEVICE
    enumerator L2TAP_SDEVICE_DRV_HNDL
    enumerator L2TAP_GDEVICE_DRV_HNDL

**WiFi default API reference**

**Header File**

- components/esp_wifi/include/esp_wifi_default.h

**Functions**

- `esp_err_t esp_netif_attach_wifi_station(esp_netif_t *esp_netif)`
  Attaches wifi station interface to supplied netif.
  
  参数 `esp_netif` – instance to attach the wifi station to
  返回 • ESP_OK on success
  • ESP_FAIL if attach failed

- `esp_err_t esp_netif_attach_wifi_ap(esp_netif_t *esp_netif)`
  Attaches wifi soft AP interface to supplied netif.
  
  参数 `esp_netif` – instance to attach the wifi AP to
  返回 • ESP_OK on success
  • ESP_FAIL if attach failed

- `esp_err_t esp_wifi_set_default_wifi_sta_handlers(void)`
  Sets default wifi event handlers for STA interface.
  
  返回 • ESP_OK on success, error returned from esp_event_handler_register if failed

- `esp_err_t esp_wifi_set_default_wifi_ap_handlers(void)`
  Sets default wifi event handlers for AP interface.
  
  返回 • ESP_OK on success, error returned from esp_event_handler_register if failed
**esp_err_t esp_wifi_clear_default_wifi_driver_and_handlers (void *esp_netif)**
Cleans default wifi event handlers for supplied network interface.

**参数** esp_netif – instance of corresponding if object

**返回**
- ESP_OK on success, error returned from esp_event_handler_register if failed

**esp_netif_t *esp_netif_create_default_wifi_ap (void)**
Creates default WIFI AP. In case of any init error this API aborts.

**备注**：The API creates esp_netif object with default WiFi access point config, attaches the netif to wifi and registers default wifi handlers.

**返回** pointer to esp-netif instance

**esp_netif_t *esp_netif_create_default_wifi_sta (void)**
Creates default WIFI STA. In case of any init error this API aborts.

**备注**：The API creates esp_netif object with default WiFi station config, attaches the netif to wifi and registers default wifi handlers.

**返回** pointer to esp-netif instance

**void esp_netif_destroy_default_wifi (void *esp_netif)**
Destroys default WIFI netif created with esp_netif_create_default_wifi…() API.

**备注**：This API unregisters wifi handlers and detaches the created object from the wifi. (this function is a no-operation if esp_netif is NULL)

**参数** esp_netif – [in] object to detach from WiFi and destroy

**esp_netif_t *esp_netif_create_wifi (wifi_interface_t wifi_if, const esp_netif_inherent_config_t *esp_netif_config)**
Creates esp_netif WiFi object based on the custom configuration.

**Attention** This API DOES NOT register default handlers!

**参数**
- wifi_if – [in] type of wifi interface
- esp_netif_config – inherent esp-netif configuration pointer

**返回** pointer to esp-netif instance

**esp_err_t esp_netif_create_default_wifi_mesh_netifs (esp_netif_t **p_netif_sta, esp_netif_t **p_netif_ap)**
Creates default STA and AP network interfaces for esp-mesh.

Both netifs are almost identical to the default station and softAP, but with DHCP client and server disabled. Please note that the DHCP client is typically enabled only if the device is promoted to a root node.

Returns created interfaces which could be ignored setting parameters to NULL if an application code does not need to save the interface instances for further processing.

**参数**
- p_netif_sta – [out] pointer where the resultant STA interface is saved (if non NULL)
• **p_netif_ap** [out] pointer where the resultant AP interface is saved (if non NULL)

ESP_OK on success

### 2.5.5 IP 网络层协议

**ESP-NETIF Custom I/O Driver**

This section outlines implementing a new I/O driver with esp-netif connection capabilities. By convention the I/O driver has to register itself as an esp-netif driver and thus holds a dependency on esp-netif component and is responsible for providing data path functions, post-attach callback and in most cases also default event handlers to define network interface actions based on driver’s lifecycle transitions.

**Packet input/output** As shown in the diagram, the following three API functions for the packet data path must be defined for connecting with esp-netif:

• `esp_netif_transmit()`
• `esp_netif_free_rx_buffer()`
• `esp_netif_receive()`

The first two functions for transmitting and freeing the rx buffer are provided as callbacks, i.e. they get called from esp-netif (and its underlying TCP/IP stack) and I/O driver provides their implementation.

The receiving function on the other hand gets called from the I/O driver, so that the driver’s code simply calls `esp_netif_receive()` on a new data received event.

**Post attach callback** A final part of the network interface initialization consists of attaching the esp-netif instance to the I/O driver, by means of calling the following API:

```c
esp_err_t esp_netif_attach(esp_netif_t *esp_netif, esp_netif_iodriver_handle...
driver_handle);
```

It is assumed that the `esp_netif_iodriver_handle` is a pointer to driver’s object, a struct derived from `struct esp_netif_driver_base_s`, so that the first member of I/O driver structure must be this base structure with pointers to

• post-attach function callback
• related esp-netif instance

As a consequence the I/O driver has to create an instance of the struct per below:

```c
typedef struct my_netif_driver_s {
    esp_netif_driver_base_t base;  /*!< base structure reserved as...
    "esp_netif_driver */
    driver_impl *h;  /*!< handle of driver...
    "implementation */
} my_netif_driver_t;
```

with actual values of `my_netif_driver_t::base.post_attach` and the actual drivers handle `my_netif_driver_t::h`. So when the `esp_netif_attach()` gets called from the initialization code, the post-attach callback from I/O driver’s code gets executed to mutually register callbacks between esp-netif and I/O driver instances. Typically the driver is started as well in the post-attach callback. An example of a simple post-attach callback is outlined below:

```c
static esp_err_t my_post_attach_start(esp_netif_t * esp_netif, void * args) {
    my_netif_driver_t *driver = args;
    const esp_netif_driver_ifconfig_t driver_ifconfig = {
        .driver_free_rx_buffer = my_free_rx_buf,
        .transmit = my_transmit,
    }
    (下页続き)
```
Default handlers

I/O drivers also typically provide default definitions of lifecycle behaviour of related network interfaces based on state transitions of I/O drivers. For example `driver start -> network start`, etc. An example of such a default handler is provided below:

```c
esp_err_t my_driver_netif_set_default_handlers(my_netif_driver_t *driver, esp_netif_t *esp_netif)
{
    driver_set_event_handler(driver->driver_impl, esp_netif_action_start, MY_DRV_,
    EVENT_START, esp_netif);
    driver_set_event_handler(driver->driver_impl, esp_netif_action_stop, MY_DRV_,
    EVENT_STOP, esp_netif);
    return ESP_OK;
}
```

Network stack connection

The packet data path functions for transmitting and freeing the rx buffer (defined in the I/O driver) are called from the esp-netif, specifically from its TCP/IP stack connecting layer.

Note, that IDF provides several network stack configurations for the most common network interfaces, such as for the WiFi station or Ethernet. These configurations are defined in `esp_netif/include/esp_netif_defaults.h` and should be sufficient for most network drivers. (In rare cases, expert users might want to define custom lwIP based interface layers; it is possible, but an explicit dependency to lwIP needs to be set)

The following API reference outlines these network stack interaction with the esp-netif:

### Header File

- `components/esp_netif/include/esp_netif_net_stack.h`

### Functions

- `esp_netif_t *esp_netif_get_handle_from_netif_impl(void *dev)`
  - Returns esp-netif handle.
  - 参数 `dev` – [in] opaque ptr to network interface of specific TCP/IP stack
  - 返回 handle to related esp-netif instance

- `void *esp_netif_get_netif_impl(esp_netif_t *esp_netif)`
  - Returns network stack specific implementation handle (if supported)
  - Note that it is not supported to acquire PPP netif impl pointer and this function will return NULL for esp_netif instances configured to PPP mode
  - 参数 `esp_netif` – [in] Handle to esp-netif instance
  - 返回 handle to related network stack netif handle

- `esp_err_t esp_netif_set_link_speed(esp_netif_t *esp_netif, uint32_t speed)`
  - Set link-speed for the specified network interface.
  - 参数
    - `esp_netif` – [in] Handle to esp-netif instance
    - `speed` – [in] Link speed in bit/s
  - 返回 ESP_OK on success
**esp_err_t esp_netif_transmit** *(esp_netif_t *esp_netif, void *data, size_t len)*

Outputs packets from the TCP/IP stack to the media to be transmitted.

This function gets called from network stack to output packets to IO driver.

- **esp_netif** – [in] Handle to esp-netif instance
- **data** – [in] Data to be transmitted
- **len** – [in] Length of the data frame

返回 ESP_OK on success, an error passed from the I/O driver otherwise

**esp_err_t esp_netif_transmit_wrap** *(esp_netif_t *esp_netif, void *data, size_t len, void *netstack_buf)*

Outputs packets from the TCP/IP stack to the media to be transmitted.

This function gets called from network stack to output packets to IO driver.

- **esp_netif** – [in] Handle to esp-netif instance
- **data** – [in] Data to be transmitted
- **len** – [in] Length of the data frame
- **netstack_buf** – [in] net stack buffer

返回 ESP_OK on success, an error passed from the I/O driver otherwise

**void esp_netif_free_rx_buffer** *(void *esp_netif, void *buffer)*

Free the rx buffer allocated by the media driver.

This function gets called from network stack when the rx buffer to be freed in IO driver context, i.e. to deallocate a buffer owned by io driver (when data packets were passed to higher levels to avoid copying)

- **esp_netif** – [in] Handle to esp-netif instance
- **buffer** – [in] Rx buffer pointer

TCP/IP 套接字 API 的示例代码放在 ESP-IDF 示例项目的 protocols/sockets 目录下。

### 2.5.6 应用层协议

应用层网络协议（IP 网络层协议之上）的相关文档放在应用层协议 目录下。

### 2.6 外设 API

#### 2.6.1 Analog to Digital Converter (ADC) Oneshot Mode Driver

**Introduction**

The Analog to Digital Converter is an on-chip sensor which is able to measure analog signals from dedicated analog IO pads.

The ADC on ESP32 can be used in scenario(s) like:

- Generate one-shot ADC conversion result
- Generate continuous ADC conversion results

This guide will introduce ADC oneshot mode conversion.
Chapter 2. API

Functional Overview

The following sections of this document cover the typical steps to install and operate an ADC:

- **Resource Allocation** - covers which parameters should be set up to get an ADC handle and how to recycle the resources when ADC finishes working.
- **Unit Configuration** - covers the parameters that should be set up to configure the ADC unit, so as to get ADC conversion raw result.
- **Read Conversion Result** - covers how to get ADC conversion raw result.
- **Hardware Limitations** - describes the ADC related hardware limitations.
- **Power Management** - covers power management related.
- **IRAM Safe** - describes tips on how to read ADC conversion raw result when cache is disabled.
- **Thread Safety** - lists which APIs are guaranteed to be thread safe by the driver.
- **Kconfig Options** - lists the supported Kconfig options that can be used to make a different effect on driver behavior.

**Resource Allocation**

The ADC oneshot mode driver is implemented based on ESP32 SAR ADC module. Different ESP chips might have different number of independent ADCs. From oneshot mode driver’s point of view, an ADC instance is represented by `adc_oneshot_unit_handle_t`.

To install an ADC instance, set up the required initial configuration structure `adc_oneshot_unit_init_cfg_t`:

- `adc_oneshot_unit_init_cfg_t::unit_id` selects the ADC. Please refer to the datasheet to know dedicated analog IOs for this ADC.
- `adc_oneshot_unit_init_cfg_t::ulp_mode` sets if the ADC will be working under ULP mode.

After setting up the initial configurations for the ADC, call `adc_oneshot_new_unit()` with the prepared `adc_oneshot_unit_init_cfg_t`. This function will return an ADC unit handle, if the allocation is successful.

This function may fail due to various errors such as invalid arguments, insufficient memory, etc. Specifically, when the to-be-allocated ADC instance is registered already, this function will return `ESP_ERR_NOT_FOUND` error. Number of available ADC(s) is recorded by `SOC_ADC_PERIPH_NUM`.

If a previously created ADC instance is no longer required, you should recycle the ADC instance by calling `adc_oneshot_del_unit()`, related hardware and software resources will be recycled as well.

**Create an ADC Unit Handle under Normal Oneshot Mode**

```c
adc_oneshot_unit_handle_t adcl_handle;
adc_oneshot_unit_init_cfg_t init_config1 = {
    .unit_id = ADC_UNIT_1,
    .ulp_mode = ADC_ULP_MODE_DISABLE,
};
ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &adcl_handle));
```

**Recycle the ADC Unit**

```c
ESP_ERROR_CHECK(adc_oneshot_del_unit(adcl_handle));
```

**Unit Configuration**

After an ADC instance is created, set up the `adc_oneshot_chan_cfg_t` to configure ADC IOs to measure analog signal:

- `adc_oneshot_chan_cfg_t::atten`, ADC attenuation. Refer to the On-Chip Sensor chapter in TRM.
- `adc_oneshot_chan_cfg_t::bitwidth`, the bitwidth of the raw conversion result.
Chapter 2. API 参考

### For the IO corresponding ADC channel number. Check datasheet to know the ADC IOs. On the other hand, `adc_continuous_io_to_channel()` and `adc_continuous_channel_to_io()` can be used to know the ADC channels and ADC IOs.

To make these settings take effect, call `adc_oneshot_config_channel()` with above configuration structure. You should specify an ADC channel to be configured as well. This function (`adc_oneshot_config_channel()`) can be called multiple times to configure different ADC channels. The Driver will save each of these channel configurations internally.

#### Configure Two ADC Channels

```c
adc_oneshot_chan_cfg_t config = {
    .bitwidth = ADC_BITWIDTH_DEFAULT,
    .atten = ADC_ATTEN_DB_11,
};
ESP_ERROR_CHECK(adc_oneshot_config_channel(adcl_handle, EXAMPLE_ADC1_CHAN0, &config));
ESP_ERROR_CHECK(adc_oneshot_config_channel(adcl_handle, EXAMPLE_ADC1_CHAN1, &config));
```

#### Read Conversion Result

After above configurations, the ADC is ready to measure the analog signal(s) from the configured ADC channel(s). Call `adc_oneshot_read()` to get the conversion raw result of an ADC channel.

- `adc_oneshot_read()` is safe to use. ADC(s) are shared by some other drivers / peripherals, see Hardware Limitations. This function uses mutexes to avoid concurrent hardware usage. Therefore, this function should not be used in an ISR context. This function may fail when the ADC is in use by other drivers / peripherals, and return `ESP_ERR_TIMEOUT`. Under this condition, the ADC raw result is invalid.

These two functions will both fail due to invalid arguments.

The ADC conversion results read from these two functions are raw data. To calculate the voltage based on the ADC raw results, this formula can be used:

\[
V_{out} = \frac{D_{out} \times V_{max}}{D_{max}} \quad (1)
\]

where:

<table>
<thead>
<tr>
<th>Vout</th>
<th>Digital output result, standing for the voltage.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dout</td>
<td>ADC raw digital reading result.</td>
</tr>
<tr>
<td>Vmax</td>
<td>Maximum measurable input analog voltage, this is related to the ADC attenuation, please refer to the On-Chip Sensor chapter in TRM.</td>
</tr>
<tr>
<td>Dmax</td>
<td>Maximum of the output ADC raw digital reading result, which is (2^{\text{bitwidth}}), where <code>bitwidth</code> is the <code>adc_oneshot_chan_cfg_t:bitwidth</code> configured before.</td>
</tr>
</tbody>
</table>

To do further calibration to convert the ADC raw result to voltage in mV, please refer to calibration doc Analog to Digital Converter (ADC) Calibration Driver.

#### Read Raw Result

```c
ESP_ERROR_CHECK(adc_oneshot_read(adcl_handle, EXAMPLE_ADC1_CHAN0, &adc_raw[0][0]));
ESP_LOGI(TAG, "ADC%d Channel[%d] Raw Data: %d", ADC_UNIT_1 + 1, EXAMPLE_ADC1_CHAN0, adc_raw[0][0]);
ESP_ERROR_CHECK(adc_oneshot_read(adcl_handle, EXAMPLE_ADC1_CHAN1, &adc_raw[0][1]));
ESP_LOGI(TAG, "ADC%d Channel[%d] Raw Data: %d", ADC_UNIT_1 + 1, EXAMPLE_ADC1_CHAN1, adc_raw[0][1]);
```
Hardware Limitations

- Random Number Generator uses ADC as a input source. When ADC `adc_oneshot_read()` works, the random number generated from RNG will be less random.
- A specific ADC unit can only work under one operating mode at any one time, either continuous mode or oneshot mode. `adc_oneshot_read()` has provided the protection.
- ADC2 is also used by the Wi-Fi. `adc_oneshot_read()` has provided the protection between Wi-Fi driver and ADC oneshot mode driver.
- ESP32 DevKitC: GPIO 0 cannot be used due to external auto program circuits.
- ESP-WROVER-KIT: GPIO 0, 2, 4 and 15 cannot be used due to external connections for different purposes.

Power Management

When power management is enabled (i.e. `CONFIG_PM_ENABLE` is on), the system clock frequency may be adjusted when the system is in an idle state. However, the ADC oneshot mode driver works in a polling routine, the `adc_oneshot_read()` will poll the CPU until the function returns. During this period of time, the task in which ADC oneshot mode driver resides won’t be blocked. Therefore the clock frequency is stable when reading.

IRAM Safe

By default, all the ADC oneshot mode driver APIs are not supposed to be run when the Cache is disabled (Cache may be disabled due to many reasons, such as Flash writing/erasing, OTA, etc.). If these APIs executes when the Cache is disabled, you will probably see errors like Illegal Instruction or Load/Store Prohibited.

Thread Safety

- `adc_oneshot_new_unit()`
- `adc_oneshot_config_channel()`
- `adc_oneshot_read()`

Above functions are guaranteed to be thread safe. Therefore, you can call them from different RTOS tasks without protection by extra locks.

- `adc_oneshot_del_unit()` is not thread safe. Besides, concurrently calling this function may result in failures of above thread-safe APIs.

Kconfig Options

- `CONFIG_ADC_ONESHOT_CTRL_FUNC_IN_IRAM` controls where to place the ADC fast read function (IRAM or Flash), see IRAM Safe for more details.

Application Examples

- ADC oneshot mode example: `peripherals/adc/oneshot_read`.

API Reference

Header File

- `components/hal/include/hal/adc_types.h`

Structures

```c
struct adc_digi_pattern_config_t

ADC digital controller pattern configuration.
```
Public Members

uint8_t atten
   Attenuation of this ADC channel.

uint8_t channel
   ADC channel.

uint8_t unit
   ADC unit.

uint8_t bit_width
   ADC output bit width.

struct adc_digi_output_data_t
   ADC digital controller (DMA mode) output data format. Used to analyze the acquired ADC (DMA) data.

备注: ESP32: Only type1 is valid. ADC2 does not support DMA mode.

备注: ESP32-S2: Member channel can be used to judge the validity of the ADC data, because the role of the arbiter may get invalid ADC data.

Public Members

uint16_t data
   ADC real output data info. Resolution: 12 bit.
   ADC real output data info. Resolution: 11 bit.

uint16_t channel
   ADC channel index info.
   ADC channel index info. For ESP32-S2: If (channel < ADC_CHANNEL_MAX), The data is valid. If (channel > ADC_CHANNEL_MAX), The data is invalid.

struct adc_digi_output_data_t::[anonymous]::[anonymous] type1
   ADC type1

uint16_t unit
   ADC unit index info. 0: ADC1; 1: ADC2.

struct adc_digi_output_data_t::[anonymous]::[anonymous] type2
   When the configured output format is 11bit.

uint16_t val
   Raw data value
Enumerations

```c
enum adc_unit_t
    ADC unit.
    Values:
```

```c
to
    ADC_UNIT_1
        SAR ADC 1.
```

```c
to
    ADC_UNIT_2
        SAR ADC 2.
```

```c
enum adc_channel_t
    ADC channels.
    Values:
```

```c
to
    ADC_CHANNEL_0
        ADC channel.
```

```c
to
    ADC_CHANNEL_1
        ADC channel.
```

```c
to
    ADC_CHANNEL_2
        ADC channel.
```

```c
to
    ADC_CHANNEL_3
        ADC channel.
```

```c
to
    ADC_CHANNEL_4
        ADC channel.
```

```c
to
    ADC_CHANNEL_5
        ADC channel.
```

```c
to
    ADC_CHANNEL_6
        ADC channel.
```

```c
to
    ADC_CHANNEL_7
        ADC channel.
```

```c
to
    ADC_CHANNEL_8
        ADC channel.
```

```c
to
    ADC_CHANNEL_9
        ADC channel.
```

```c
enum adc_atten_t
    ADC attenuation parameter. Different parameters determine the range of the ADC.
    Values:
```

```c
enumerator ADC_ATTEN_DB_0
    No input attenuation, ADC can measure up to approx.

enumerator ADC_ATTEN_DB_2_5
    The input voltage of ADC will be attenuated extending the range of measurement by about 2.5 dB (1.33 x)

enumerator ADC_ATTEN_DB_6
    The input voltage of ADC will be attenuated extending the range of measurement by about 6 dB (2 x)

enumerator ADC_ATTEN_DB_11
    The input voltage of ADC will be attenuated extending the range of measurement by about 11 dB (3.55 x)

enum adc_bitwidth_t
    Values:

enumerator ADC_BITWIDTH_DEFAULT
    Default ADC output bits, max supported width will be selected.

enumerator ADC_BITWIDTH_9
    ADC output width is 9Bit.

enumerator ADC_BITWIDTH_10
    ADC output width is 10Bit.

enumerator ADC_BITWIDTH_11
    ADC output width is 11Bit.

enumerator ADC_BITWIDTH_12
    ADC output width is 12Bit.

enumerator ADC_BITWIDTH_13
    ADC output width is 13Bit.

enum adc_ulp_mode_t
    Values:

enumerator ADC_ULP_MODE_DISABLE
    ADC ULP mode is disabled.

enumerator ADC_ULP_MODE_FSM
    ADC is controlled by ULP FSM.

enumerator ADC_ULP_MODE_RISCV
    ADC is controlled by ULP RISCV.
enum `adc_digi_convert_mode_t`
ADC digital controller (DMA mode) work mode.

Values:

enumerator `ADC_CONV_SINGLE_UNIT_1`
Only use ADC1 for conversion.

enumerator `ADC_CONV_SINGLE_UNIT_2`
Only use ADC2 for conversion.

enumerator `ADC_CONV_BOTH_UNIT`
Use Both ADC1 and ADC2 for conversion simultaneously.

enumerator `ADC_CONV_ALTER_UNIT`
Use both ADC1 and ADC2 for conversion by turn. e.g. ADC1 -> ADC2 -> ADC1 -> ADC2 ...

enum `adc_digi_output_format_t`
ADC digital controller (DMA mode) output data format option.

Values:

enumerator `ADC_DIGI_OUTPUT_FORMAT_TYPE1`
See `adc_digi_output_data_t.type1`

enumerator `ADC_DIGI_OUTPUT_FORMAT_TYPE2`
See `adc_digi_output_data_t.type2`

Header File
- components/esp_adc/include/esp_adc/adc_oneshot.h

Functions

```c
esp_err_t adc_oneshot_new_unit(const adc_oneshot_unit_init_cfg_t *init_config,
                                adc_oneshot_unit_handle_t *ret_unit)
```
Create a handle to a specific ADC unit.

备注: This API is thread-safe. For more details, see ADC programming guide

参数
- `init_config` [in] Driver initial configurations
- `ret_unit` [out] ADC unit handle

返回
- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_NO_MEM: No memory
- ESP_ERR_NOT_FOUND: The ADC peripheral to be claimed is already in use

```c
esp_err_t adc_oneshot_config_channel(const adc_oneshot_unit_handle_t *handle,
                                     adc_channel_t channel,
                                     const adc_oneshot_chan_cfg_t *config)
```
Set ADC oneshot mode required configurations.

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>[in] ADC handle</td>
</tr>
<tr>
<td>channel</td>
<td>[in] ADC channel to be configured</td>
</tr>
<tr>
<td>config</td>
<td>[in] ADC configurations</td>
</tr>
</tbody>
</table>

**函数**

```c
esp_err_t adc_oneshot_read(adc_oneshot_unit_handle_t handle, adc_channel_t chan, int *out_raw)
```

Get one ADC conversion raw result.

**备注:** This API is thread-safe. For more details, see ADC programming guide

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>[in] ADC handle</td>
</tr>
<tr>
<td>chan</td>
<td>[in] ADC channel</td>
</tr>
<tr>
<td>out_raw</td>
<td>[out] ADC conversion raw result</td>
</tr>
</tbody>
</table>

**返回**

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_TIMEOUT: Timeout, the ADC result is invalid

```c
esp_err_t adc_oneshot_del_unit(adc_oneshot_unit_handle_t handle)
```

Delete the ADC unit handle.

**备注:** This API is thread-safe. For more details, see ADC programming guide

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>[in] ADC handle</td>
</tr>
</tbody>
</table>

**返回**

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_NOT_FOUND: The ADC peripheral to be disclaimed isn’t in use

```c
esp_err_t adc_oneshot_io_to_channel(int io_num, adc_unit_t *unit_id, adc_channel_t *channel)
```

Get ADC channel from the given GPIO number.

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_num</td>
<td>[in] GPIO number</td>
</tr>
<tr>
<td>unit_id</td>
<td>[out] ADC unit</td>
</tr>
<tr>
<td>channel</td>
<td>[out] ADC channel</td>
</tr>
</tbody>
</table>

**返回**

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_FOUND: The IO is not a valid ADC pad
```c
esp_err_t adc_oneshot_channel_to_io(adc_unit_t unit_id, adc_channel_t channel, int* io_num)
```

Get GPIO number from the given ADC channel.

**Parameters**
- `unit_id` - [in] ADC unit
- `channel` - [in] ADC channel
- `io_num` - [out] GPIO number
- - ESP_OK: On success
- - ESP_ERR_INVALID_ARG: Invalid argument

**Structures**

struct `adc_oneshot_unit_init_cfg_t`

ADC oneshot driver initial configurations.

**Public Members**

- `adc_unit_t unit_id`
  
  ADC unit.

- `adc_ulp_mode_t ulp_mode`
  
  ADC controlled by ULP, see `adc_ulp_mode_t` struct

struct `adc_oneshot_chan_cfg_t`

ADC channel configurations.

**Public Members**

- `adc_atten_t atten`
  
  ADC attenuation.

- `adc_bitwidth_t bitwidth`
  
  ADC conversion result bits.

**Type Definitions**

typedef struct adc_oneshot_unit_ctx_t *`adc_oneshot_unit_handle_t`

Type of ADC unit handle for oneshot mode.

### 2.6.2 Analog to Digital Converter (ADC) Continuous Mode Driver

**Introduction**

The Analog to Digital Converter is an on-chip sensor which is able to measure analog signals from specific analog IO pads.

The ADC on ESP32 can be used in scenario(s) like:
- Generate one-shot ADC conversion result
- Generate continuous ADC conversion results

This guide will introduce ADC continuous mode conversion.
Driver Concepts ADC continuous mode conversion is made up with multiple Conversion Frames.

- **Conversion Frame**: One Conversion Frame contains multiple Conversion Results. Conversion Frame size is configured in `adc_continuous_new_handle()`, in bytes.
- **Conversion Result**: One Conversion Result contains multiple bytes (see `SOC_ADC_DIGI_RESULT_BYTES`). Its structure is `adc_digi_output_data_t`, including ADC unit, ADC channel and raw data.

Functional Overview

The following sections of this document cover the typical steps to install the ADC continuous mode driver, and read ADC conversion results from group of ADC channels continuously:

- **Resource Allocation** - covers which parameters should be set up to initialize the ADC continuous mode driver and how to deinitialize it.
- **ADC Configurations** - describes how to configure the ADC(s) to make it work under continuous mode.
- **ADC Control** - describes ADC control functions.
- **Register Event Callbacks** - describes how to hook users specific code to an ADC continuous mode event callback function.
- **Read Conversion Result** - covers how to get ADC conversion result.
- **Hardware Limitations** - describes the ADC related hardware limitations.
- **Power Management** - covers power management related.
- **IRAM Safe** - covers the IRAM safe functions.
- **Thread Safety** - lists which APIs are guaranteed to be thread safe by the driver.

Resource Allocation The ADC continuous mode driver is implemented based on ESP32 SAR ADC module. Different ESP targets might have different number of independent ADCs.

To create an ADC continuous mode driver handle, set up the required configuration structure `adc_continuous_handle_cfg_t`:

- `adc_continuous_handle_cfg_t::max_store_buf_size` set the maximum size (in bytes) of the pool that the driver saves ADC conversion result into. If this pool is full, new conversion results will be lost.
- `adc_continuous_handle_cfg_t::conv_frame_size` set the size of the ADC conversion frame, in bytes.

After setting up above configurations for the ADC, call `adc_continuous_new_handle()` with the prepared `adc_continuous_handle_cfg_t`. This function may fail due to various errors such as invalid arguments, insufficient memory, etc.

Especially, when this function returns `ESP_ERR_NOT_FOUND`, this means the I2S0 peripheral is in use. See Hardware Limitations for more information.

If the ADC continuous mode driver is no longer used, you should deinitialize the driver by calling `adc_continuous_deinit()`.

Initialize the ADC Continuous Mode Driver
adc_continuous_handle_cfg_t
.
adc_config = {
    .max_store_buf_size = 1024,
    .conv_frame_size = 100,
};
ESP_ERROR_CHECK(adc_continuous_new_handle(&adc_config));

Recycle the ADC Unit
ESP_ERROR_CHECK(adc_continuous_deinit());

ADC Configurations After the ADC continuous mode driver is initialized, set up the adc_continuous_config_t to configure ADC IOs to measure analog signal:

- **adc_continuous_config_t::pattern_num**, number of ADC channels that will be used.
- **adc_continuous_config_t::adc_pattern**, list of configs for each ADC channel that will be used, see below description.
- **adc_continuous_config_t::sample_freg_hz**, expected ADC sampling frequency in Hz.
- **adc_continuous_config_t::conv_mode**, continuous conversion mode.
- **adc_continuous_config_t::format**, conversion output format.

For **adc_digi_pattern_config_t**:

- **adc_digi_pattern_config_t::atten**, ADC attenuation. Refer to the On-Chip Sensor chapter in TRM.
- **adc_digi_pattern_config_t::channel**, the IO corresponding ADC channel number. See below note.
- **adc_digi_pattern_config_t::unit**, the ADC that the IO is subordinate to.
- **adc_digi_pattern_config_t::bit_width**, the bitwidth of the raw conversion result.

备注: For the IO corresponding ADC channel number. Check datasheet to acquire the ADC IOs. On the other hand, **adc_continuous_io_to_channel()** and **adc_continuous_channel_to_io()** can be used to acquire the ADC channels and ADC IOs.

To make these settings take effect, call **adc_continuous_config()** with the configuration structure above. This API may fail due to reasons like ESP_ERR_INVALID_ARG. When it returns ESP_ERR_INVALID_STATE, this means the ADC continuous mode driver is started, you shouldn’t call this API at this moment.

See ADC continuous mode example peripherals/adc/continuous_read to see configuration codes.

ADC Control

Start and Stop Calling **adc_continuous_start()** will make the ADC start to measure analog signals from the configured ADC channels, and generate the conversion results. On the contrary, calling **adc_continuous_stop()** will stop the ADC conversion.

Register Event Callbacks By calling **adc_continuous_register_event_callbacks()**, you can hook your own function to the driver ISR. Supported event callbacks are listed in adc_continuous_evt_cbs_t:

- **adc_continuous_evt_cbs_t::on_conv_done**, this is invoked when one conversion frame finishes.
- **adc_continuous_evt_cbs_t::on_pool_ovf**, this is invoked when internal pool is full. Newer conversion results will be discarded.

As above callbacks are called in an ISR context, you should always ensure the callback function is suitable for an ISR context. Blocking logics should not appear in these callbacks. Callback function prototype is declared in **adc_continuous_callback_t**.

Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151
You can also register your own context when calling `adc_continuous_register_event_callbacks()`, by the parameter `user_data`. This user data will be passed to the callback functions directly.

This function may fail due to reasons like `ESP_ERR_INVALID_ARG`. Specially, when `CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE` is enabled, this error may indicate that the callback functions aren’t in internal RAM. Check error log to know this. Besides, when it fails due to `ESP_ERR_INVALID_STATE`, this means the ADC continuous mode driver is started, you shouldn’t add callback at this moment.

**Conversion Done Event** The driver will fill in the event data of a `adc_continuous_evt_cbs_t::on_conv_done` event. Event data contains a buffer pointer to a conversion frame buffer, together with the size. Refer to `adc_continuous_evt_data_t` to know the event data structure.

**Pool Overflow Event** The ADC continuous mode driver has an internal pool to save the conversion results. When the pool is full, a pool overflow event will emerge. Under this condition, the driver won’t fill in the event data. This usually happens the speed to read data from the pool (by calling `adc_continuous_read()`) is much slower than the ADC conversion speed.

**Read Conversion Result** After calling `adc_continuous_start()`, the ADC continuous conversion starts. Call `adc_continuous_read()` to get the conversion results of the ADC channels. You need to provide a buffer to get the raw results.

This function will try to read the expected length of conversion results each time.

- If the requested length isn’t reached, the function will still move the data from the internal pool to the buffer you prepared. Therefore, check the `out_length` to know the actual size of conversion results.
- If there is no conversion result generated in the internal pool, the function will block for `timeout_ms` until the conversion results are generated. If there is still no generated results, the function will return `ESP_ERR_TIMEOUT`.
- If the generated results fill up the internal pool, new generated results will be lost. Next time when the `adc_continuous_read()` is called, this function will return `ESP_ERR_INVALID_STATE` indicating this situation.

This API aims to give you a chance to read all the ADC continuous conversion results.

The ADC conversion results read from above function are raw data. To calculate the voltage based on the ADC raw results, this formula can be used:

\[
V_{out} = D_{out} \times \frac{V_{max}}{D_{max}} \quad (1)
\]

where:
- `V_{out}`: Digital output result, standing for the voltage.
- `D_{out}`: ADC raw digital reading result.
- `V_{max}`: Maximum measurable input analog voltage, this is related to the ADC attenuation, please refer to the On-Chip Sensor chapter in TRM.
- `D_{max}`: Maximum of the output ADC raw digital reading result, which is \(2^\text{bitwidth}\), where bitwidth is the `::cpp::member::adc_digi_pattern_config_t::bit_width` configured before.
To do further calibration to convert the ADC raw result to voltage in mV, please refer to calibration doc *Analog to Digital Converter (ADC) Calibration Driver*.

**Hardware Limitations**

- A specific ADC unit can only work under one operating mode at any one time, either continuous mode or oneshot mode. *adc_continuous_start()* has provided the protection.
- Random Number Generator uses ADC as an input source. When ADC continuous mode driver works, the random number generated from RNG will be less random.
- ADC2 is also used by the Wi-Fi. *adc_continuous_start()* has provided the protection between Wi-Fi driver and ADC continuous mode driver.
- ADC continuous mode driver uses I2S0 peripheral as hardware DMA fifo. Therefore, if I2S0 is in use already, *adc_continuous_new_handle()* will return ESP_ERR_NOT_FOUND.
- ESP32 DevKitC: GPIO 0 cannot be used due to external auto program circuits.
- ESP-WROVER-KIT: GPIO 0, 2, 4 and 15 cannot be used due to external connections for different purposes.

**Power Management**

When power management is enabled (i.e. *CONFIG_PM_ENABLE* is on), the APB clock frequency may be adjusted when the system is in an idle state, thus potentially changing the behavior of ADC continuous conversion.

However, the continuous mode driver can prevent this change by acquiring a power management lock of type *ESP_PM_APB_FREQ_MAX*. The lock is acquired after the continuous conversion is started by *adc_continuous_start()*). Similarly, the lock will be released after *adc_continuous_stop()*). Therefore, *adc_continuous_start()* and *adc_continuous_stop()* should appear in pairs, otherwise the power management will be out of action.

**IRAM Safe**

All the ADC continuous mode driver APIs are not IRAM-safe. They are not supposed to be run when the Cache is disabled. By enabling the Kconfig option *CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE*, driver internal ISR handler is IRAM-safe, which means even when the Cache is disabled, the driver will still save the conversion results into its internal pool.

**Thread Safety**

ADC continuous mode driver APIs are not guaranteed to be thread safe. However, the share hardware mutual exclusion is provided by the driver. See *Hardware Limitations* for more details.

**Application Examples**

- ADC continuous mode example: *peripherals/adc/continuous_read*.

**API Reference**

**Header File**

- *components/esp_adc/include/esp_adc/adc_continuous.h*

**Functions**

```c
esp_err_t adc_continuous_new_handle (const adc_continuous_handle_cfg_t *hdl_config,
                                  adc_continuous_handle_t *ret_handle)
```

Initialize ADC continuous driver and get a handle to it.

参数

- **hdl_config** - [in] Pointer to ADC initilization config. Refer to *adc_continuous_handle_cfg_t*.
- **ret_handle** - [out] ADC continuous mode driver handle

返回

Espressif Systems 972 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
• ESP_ERR_INVALID_ARG If the combination of arguments is invalid.
• ESP_ERR_NOT_FOUND No free interrupt found with the specified flags
• ESP_ERR_NO_MEM If out of memory
• ESP_OK On success

**esp_err_t adc_continuous_config**(adc_continuous_handle_t handle, const adc_continuous_config_t *config)

Set ADC continuous mode required configurations.

**参数**
- handle [in] ADC continuous mode driver handle
- config [in] Refer to adc_digi_config_t.

**返回**
- ESP_ERR_INVALID_STATE: Driver state is invalid, you shouldn’t call this API at this moment
- ESP_ERR_INVALID_ARG: If the combination of arguments is invalid.
- ESP_OK: On success

**esp_err_t adc_continuous_register_event_callbacks**(adc_continuous_handle_t handle, const adc_continuous_evt_cbs_t *cbs, void *user_data)

Register callbacks.

**备注**: User can deregister a previously registered callback by calling this function and setting the to-be-deregistered callback member int the cbs structure to NULL.

**备注**: When CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. Involved variables (including user_data) should be in internal RAM as well.

**备注**: You should only call this API when the ADC continuous mode driver isn’t started. Check return value to know this.

**参数**
- handle [in] ADC continuous mode driver handle
- cbs [in] Group of callback functions
- user_data [in] User data, which will be delivered to the callback functions directly

**返回**
- ESP_OK: On success
- ESP_ERR_INVALID_STATE: Driver state is invalid.
- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_ERR_INVALID_STATE: Driver state is invalid, you shouldn’t call this API at this moment

**esp_err_t adc_continuous_start**(adc_continuous_handle_t handle)

Start the ADC under continuous mode. After this, the hardware starts working.

**参数**
- handle [in] ADC continuous mode driver handle

**返回**
- ESP_ERR_INVALID_STATE: Driver state is invalid.
- ESP_OK: On success

**esp_err_t adc_continuous_read**(adc_continuous_handle_t handle, uint8_t *buf, uint32_t length_max, uint32_t *out_length, uint32_t timeout_ms)

Read bytes from ADC under continuous mode.

**参数**
Chapter 2. API

- **handle** - [in] ADC continuous mode driver handle
- **buf** - [out] Conversion result buffer to read from ADC. Suggest convert to `adc_digi_output_data_t` for ADC Conversion Results. See @brief Driver Backgrounds to know this concept.
- **length_max** - [in] Expected length of the Conversion Results read from the ADC, in bytes.
- **out_length** - [out] Real length of the Conversion Results read from the ADC via this API, in bytes.
- **timeout_ms** - [in] Time to wait for data via this API, in millisecond.

### esp_err_t adc_continuous_stop (adc_continuous_handle_t handle)
Stop the ADC. After this, the hardware stops working.

- 参数 **handle** - [in] ADC continuous mode driver handle
- 返回
  - ESP_ERR_INVALID_STATE: Driver state is invalid. Usually it means the ADC sampling rate is faster than the task processing rate.
  - ESP_ERR_TIMEOUT: Operation timed out
  - ESP_OK: On success

### esp_err_t adc_continuous_deinit (adc_continuous_handle_t handle)
Deinitialize the ADC continuous driver.

- 参数 **handle** - [in] ADC continuous mode driver handle
- 返回
  - ESP_ERR_INVALID_STATE: Driver state is invalid.
  - ESP_OK: On success

### esp_err_t adc_continuous_io_to_channel (int io_num, adc_unit_t *unit_id, adc_channel_t *channel)
Get ADC channel from the given GPIO number.

- 参数
  - **io_num** - [in] GPIO number
  - **unit_id** - [out] ADC unit
  - **channel** - [out] ADC channel
- 返回
  - ESP_OK: On success
  - ESP_ERR_INVALID_ARG: Invalid argument
  - ESP_ERR_NOT_FOUND: The IO is not a valid ADC pad

### esp_err_t adc_continuous_channel_to_io (adc_unit_t unit_id, adc_channel_t channel, int *io_num)
Get GPIO number from the given ADC channel.

- 参数
  - **unit_id** - [in] ADC unit
  - **channel** - [in] ADC channel
  - **io_num** - [out] GPIO number
  - ESP_OK: On success
  - ESP_ERR_INVALID_ARG: Invalid argument

### Structures

**struct adc_continuous_handle_cfg_t**
ADC continuous mode driver initial configurations.

### Public Members
uint32_t max_store_buf_size
Max length of the conversion Results that driver can store, in bytes.

uint32_t conv_frame_size
Conversion frame size, in bytes. This should be in multiples of
SOC_ADC_DIGI_DATA_BYTES_PER_CONV.

struct adc_continuous_config_t
ADC continuous mode driver configurations.

Public Members

uint32_t pattern_num
Number of ADC channels that will be used.

adc_digi_pattern_config_t *adc_pattern
List of configs for each ADC channel that will be used.

uint32_t sample_freq_hz
The expected ADC sampling frequency in Hz. Please refer to soc/soc_caps.h to know available
sampling frequency range

adc_digi_convert_mode_t conv_mode
ADC DMA conversion mode, see adc_digi_convert_mode_t.

adc_digi_output_format_t format
ADC DMA conversion output format, see adc_digi_output_format_t.

struct adc_continuous_evt_data_t
Event data structure.

The conv_frame_buffer is maintained by the driver itself, so never free this piece of memory.

Public Members

uint8_t *conv_frame_buffer
Pointer to conversion result buffer for one conversion frame.

uint32_t size
Conversion frame size.

struct adc_continuous_evt_cbs_t
Group of ADC continuous mode callbacks.

These callbacks are all running in an ISR environment.
When CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. Involved variables should be in internal RAM as well.

Public Members

```c
void adc_continuous_callback_t on_conv_done(adc_continuous_handle_t handle, const adc_continuous_evt_data_t *edata, void *user_data);
```

- Event callback, invoked when one conversion frame is done. See @brief Driver Backgrounds to know conversion frame concept.

```c
void adc_continuous_callback_t on_pool_ovf(adc_continuous_handle_t handle, const adc_continuous_evt_data_t *edata, void *user_data);
```

- Event callback, invoked when the internal pool is full.

Macros

```c
#define ADC_MAX_DELAY Driver Backgrounds
```

Type Definitions

```c
typedef struct adc_continuous_ctx_t *adc_continuous_handle_t
```

- Type of adc continuous mode driver handle.

```c
typedef bool (*adc_continuous_callback_t)(adc_continuous_handle_t handle, const adc_continuous_evt_data_t *edata, void *user_data);
```

- Prototype of ADC continuous mode event callback.

Function Parameters:
- `handle` [in] ADC continuous mode driver handle
- `edata` [in] Pointer to ADC continuous mode event data
- `user_data` [in] User registered context, registered when in adc_continuous_register_event_callbacks()

Return: Whether a high priority task is woken up by this function

2.6.3 Analog to Digital Converter (ADC) Calibration Driver

Introduction

Based on series of comparisons with the reference voltage, ESP32 ADC determines each bit of the output digital result. Per design the ESP32 ADC reference voltage is 1100 mV, however the true reference voltage can range from 1000 mV to 1200 mV among different chips. This guide will introduce an ADC calibration driver to minimize this difference.

Functional Overview

The following sections of this document cover the typical steps to install and use the ADC calibration driver:

- **Calibration Scheme Creation** - covers how to create a calibration scheme handle and delete the calibration scheme handle.
- **Calibration Configuration** - covers how to configure the calibration driver to calculate necessary characteristics used for calibration.
- **Result Conversion** - covers how to convert ADC raw result to calibrated result.
- **Thread Safety** - lists which APIs are guaranteed to be thread safe by the driver.
- **Minimize Noise** - describes a general way to minimize the noise.
**Kconfig Options** - lists the supported Kconfig options that can be used to make a different effect on driver behavior.

**Calibration Scheme Creation** The ADC calibration driver provides ADC calibration scheme(s). From calibration driver’s point of view, an ADC calibration scheme is created to an ADC calibration handle `adc_cal_handle_t`. `adc_cali_check_scheme()` can be used to know which calibration scheme is supported on the chip. For those users who are already aware of the supported scheme, this step can be skipped. Just call the corresponding function to create the scheme handle.

For those users who use their custom ADC calibration schemes, you could either modify this function `adc_cali_check_scheme()`, or just skip this step and call your custom creation function.

**ADC Calibration Line Fitting Scheme** ESP32 supports `ADC_CALI_SCHEME_VER_LINE_FITTING` scheme. To create this scheme, set up `adc_cali_line_fitting_config_t` first.

- `adc_cali_line_fitting_config_t::unit_id`, the ADC that your ADC raw results are from.
- `adc_cali_line_fitting_config_t::atten`, ADC attenuation that your ADC raw results use.
- `adc_cali_line_fitting_config_t::bitwidth`, the ADC raw result bitwidth.

There is also a configuration `adc_cali_line_fitting_config_t::default_vref`. Normally this can be simply set to 0. Line Fitting scheme doesn’t rely on this value. However, if the Line Fitting scheme required eFuse bits are not burnt on your board, driver will rely on this value to do the calibration.

You can use `adc_cali_scheme_line_fitting_check_efuse()` to check the eFuse bits. Normally the Line Fitting scheme eFuse value will be `ADC_CALI_LINE_FITTING_EFUSE_VAL_EFUSE_TP` or `ADC_CALI_LINE_FITTING_EFUSE_VAL_EFUSE_VREF`. This means Line Fitting scheme will use calibration parameters burnt in the eFuse to do the calibration.

When the Line Fitting scheme eFuse value is `ADC_CALI_LINE_FITTING_EFUSE_VAL_DEFAULT_VREF`, you need to set the `esp_adc_cali_line_fitting_init::default_vref`. Default vref is an estimate of the ADC reference voltage provided by the users as a parameter during calibration.

After setting up the configuration structure, call `adc_cali_create_scheme_line_fitting()` to create a Line Fitting calibration scheme handle.

```c
ESP_LOGI(TAG, "calibration scheme version is %s", "Line Fitting");
adc_cali_line_fitting_config_t cali_config = {
    .unit_id = unit,
    .atten = atten,
    .bitwidth = ADC_BITWIDTH_DEFAULT,
};
ESP_ERROR_CHECK(adc_cali_create_scheme_line_fitting(&cali_config, &handle));
```

When the ADC calibration is no longer used, please delete the calibration scheme handle by calling `adc_cali_delete_scheme_line_fitting()`.

**Delete Line Fitting Scheme**

```c
ESP_LOGI(TAG, "delete %s calibration scheme", "Line Fitting");
ESP_ERROR_CHECK(adc_cali_delete_scheme_line_fitting(handle));
```

**备注:** For users who want to use their custom calibration schemes, you could provide a creation function to create your calibration scheme handle. Check the function table `adc_cali_scheme_t` in components/esp_adc/interface/adc_cali_interface.h to know the ESP ADC calibration interface.
Result Conversion After setting up the calibration characteristics, you can call `adc_cali_raw_to_voltage()` to convert the ADC raw result into calibrated result. The calibrated result is in the unit of mV. This function may fail due to invalid argument. Especially, if this function returns `ESP_ERR_INVALID_STATE`, this means the calibration scheme isn’t created. You need to create a calibration scheme handle, use `adc_cali_check_scheme()` to know the supported calibration scheme. On the other hand, you could also provide a custom calibration scheme and create the handle.

Get Voltage

```c
ESP_ERROR_CHECK(adc_cali_raw_to_voltage(adc_cali_handle, adc_raw[0][0], &voltage[0][0]));
ESP_LOGI(TAG, "ADC%d Channel[%d] Cali Voltage: %d mV", ADC_UNIT_1 + 1, EXAMPLE_adc1_CHAN0, voltage[0][0]);
```

Thread Safety The factory function `esp_adc_cali_new_scheme()` is guaranteed to be thread safe by the driver. Therefore, you can call them from different RTOS tasks without protection by extra locks. Other functions that take the `adc_cali_handle_t` as the first positional parameter are not thread safe, you should avoid calling them from multiple tasks.

Kconfig Options

- `CONFIG_ADC_CAL_EFUSE_TP_ENABLE`, disable this to decrease the code size, if you are aware of the calibration eFuse value ADC_CALI_LINE_FITTING_EFUSE_VAL_EFUSE_TP isn’t this one.
- `CONFIG_ADC_CAL_EFUSE_VREF_ENABLE`, disable this to decrease the code size, if you are aware of the calibration eFuse value ADC_CALI_LINE_FITTING_EFUSE_VAL_EFUSE_VREF isn’t this one.
- `CONFIG_ADC_CAL_LUT_ENABLE`, disable this to decrease the code size, if you don’t calibrate the ADC raw results under ADC_ATTEN_DB_11.

Minimize Noise The ESP32 ADC can be sensitive to noise leading to large discrepancies in ADC readings. Depending on the usage scenario, you may need to connect a bypass capacitor (e.g. a 100 nF ceramic capacitor) to the ADC input pad in use, to minimize noise. Besides, multisampling may also be used to further mitigate the effects of noise.

API Reference

Header File

- `components/esp_adc/include/esp_adc/adc_cali.h`

Functions

```c
esp_err_t adc_cali_check_scheme(adc_cali_scheme_ver_t *scheme_mask)
```

Check the supported ADC calibration scheme.

参数 `scheme_mask` -[out] Supported ADC calibration scheme(s)

返回

- ESP_OK: On success
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_NOT_SUPPORTED: No supported calibration scheme

```c
esp_err_t adc_cali_raw_to_voltage(adc_cali_handle_t handle, int raw, int *voltage)
```

Convert ADC raw data to calibrated voltage.

参数

- `handle` -[in] ADC calibration handle
- `raw` -[in] ADC raw data
- `voltage` -[out] Calibrated ADC voltage (in mV)
图 6: Graph illustrating noise mitigation using capacitor and multisampling of 64 samples.

返回
• ESP_OK: On success
• ESP_ERR_INVALID_ARG: Invalid argument
• ESP_ERR_INVALID_STATE: Invalid state, scheme didn’t registered

Type Definitions
typedef struct adc_cali_scheme_t *adc_cali_handle_t
ADC calibration handle.

Enumerations
enum adc_cali_scheme_ver_t
ADC calibration scheme.
Values:

enumerator ADC_CALI_SCHEME_VER_LINE_FITTING
Line fitting scheme.

enumerator ADC_CALI_SCHEME_VER_CURVE_FITTING
Curve fitting scheme.

Header File
• components/esp_adc/include/esp_adc/adc_cali_scheme.h
Chapter 2. API

2.6.4 Clock Tree

This section lists definitions of the ESP32’s supported root clocks and module clocks. These definitions are commonly used in the driver configuration, to help user select a proper source clock for the peripheral.

Root Clocks

Root clocks generate reliable clock signals. These clock signals then pass through various gates, muxes, dividers, or multipliers to become the clock sources for every functional module: the CPU core(s), WIFI, BT, the RTC, and the peripherals.

ESP32’s root clocks are listed in \texttt{soc_root_clk_t}:

- **Internal 8MHz RC Oscillator (RC_FAST)**
  This RC oscillator generates a \(\sim 8.5\)MHz clock signal output as the RC_FAST_CLK. The \(\sim 8.5\)MHz signal output is also passed into a configurable divider, which by default divides the input clock frequency by 256, to generate a RC_FAST_D256_CLK. The exact frequency of RC_FAST_CLK can be computed in runtime through calibration on the RC_FAST_D256_CLK.

- **External 2~40MHz Crystal (XTAL)**

- **Internal 150kHz RC Oscillator (RC_SLOW)**
  This RC oscillator generates a \(\sim 150\)kHz clock signal output as the RC_SLOW_CLK. The exact frequency of this clock can be computed in runtime through calibration.

- **External 32kHz Crystal - optional (XTAL32K)**
  The clock source for this XTAL32K_CLK can be either a 32kHz crystal connecting to the 32K_XP and 32K_XN pins or a 32kHz clock signal generated by an external circuit. The external signal must be connected to the 32K_XN pin. Additionally, a 1nF capacitor must be placed between the 32K_XP pin and ground. In this case, the 32K_XP pin cannot be used as a GPIO pin. XTAL32K_CLK can also be calibrated to get its exact frequency.

Typically, the frequency of the signal generated from a RC oscillator circuit is less accurate and more sensitive to environment comparing to the signal generated from a crystal. ESP32 provides several clock source options for the RTC_SLOW_CLK, and users can make the choice based on the requirements for system time accuracy and power consumption (refer to \texttt{RTC} for more details).

Module Clocks

ESP32’s available module clocks are listed in \texttt{soc_module_clk_t}. Each module clock has a unique ID. You can get more information on each clock by checking the documented enum value.

API Reference

Header File

- \texttt{components/soc/esp32/include/soc/clk_tree_defs.h}

Macros

\texttt{SOC_CLK_RC_FAST_FREQ_APPROX}

Approximate RC_FAST_CLK frequency in Hz

\texttt{SOC_CLK_RC_SLOW_FREQ_APPROX}

Approximate RC_SLOW_CLK frequency in Hz
**Chapter 2. API**

**SOC_CLK_RC_FAST_D256_FREQ_APPROX**
Approximate RC_FAST_D256_CLK frequency in Hz

**SOC_CLK_XTAL32K_FREQ_APPROX**
Approximate XTAL32K_CLK frequency in Hz

**SOC_GPTIMER_CLKS**
Array initializer for all supported clock sources of GPTimer.
The following code can be used to iterate all possible clocks:

```c
soc_periph_gptimer_clk_src_t gptimer_clks[] = (soc_periph_gptimer_clk_src_t)SOC_GPTIMER_CLKS;
for (size_t i = 0; i < sizeof(gptimer_clks) / sizeof(gptimer_clks[0]); i++) {
    soc_periph_gptimer_clk_src_t clk = gptimer_clks[i];
    // Test GPTimer with the clock `clk`
}
```

**SOC_LCD_CLKS**
Array initializer for all supported clock sources of LCD.

**SOC_RMT_CLKS**
Array initializer for all supported clock sources of RMT.

**SOC_MCPWM_TIMER_CLKS**
Array initializer for all supported clock sources of MCPWM Timer.

**SOC_MCPWM_CAPTURE_CLKS**
Array initializer for all supported clock sources of MCPWM Capture Timer.

**SOC_I2S_CLKS**
Array initializer for all supported clock sources of I2S.

**SOC_I2C_CLKS**
Array initializer for all supported clock sources of I2C.

**SOC_SDM_CLKS**
Array initializer for all supported clock sources of SDM.

**SOC_DAC_DIGI_CLKS**
Array initializer for all supported clock sources of DAC digital controller.

**SOC_DAC_COSINE_CLKS**
Array initializer for all supported clock sources of DAC cosine wave generator.

**SOC_TWAI_CLKS**
Array initializer for all supported clock sources of TWAI.

**Enumerations**
enum soc_root_clk_t
    Root clock.
    Values:

    enumerator SOC_ROOT_CLK_INT_RC_FAST
        Internal 8MHz RC oscillator
    enumerator SOC_ROOT_CLK_INT_RC_SLOW
        Internal 150kHz RC oscillator
    enumerator SOC_ROOT_CLK_EXT_XTAL
        External 2~40MHz crystal
    enumerator SOC_ROOT_CLK_EXT_XTAL32K
        External 32kHz crystal/clock signal

enum soc_cpu_clk_src_t
    CPU_CLK mux inputs, which are the supported clock sources for the CPU_CLK.
    Values:

    enumerator SOC_CPU_CLK_SRC_XTAL
        Select XTAL_CLK as CPU_CLK source
    enumerator SOC_CPU_CLK_SRC_PLL
        Select PLL_CLK as CPU_CLK source (PLL_CLK is the output of 40MHz crystal oscillator frequency multiplier, can be 480MHz or 320MHz)
    enumerator SOC_CPU_CLK_SRC_RC_FAST
        Select RC_FAST_CLK as CPU_CLK source
    enumerator SOC_CPU_CLK_SRC_APLL
        Select APLL_CLK as CPU_CLK source
    enumerator SOC_CPU_CLK_SRC_INVALID
        Invalid CPU_CLK source

enum soc_rtc_slow_clk_src_t
    RTC_SLOW_CLK mux inputs, which are the supported clock sources for the RTC_SLOW_CLK.
    Values:

    enumerator SOC_RTC_SLOW_CLK_SRC_RC_SLOW
        Select RC_SLOW_CLK as RTC_SLOW_CLK source
enumerator **SOC_RTC_SLOW_CLK_SRC_XTAL32K**
Select XTAL32K_CLK as RTC_SLOW_CLK source

enumerator **SOC_RTC_SLOW_CLK_SRC_RC_FAST_D256**
Select RC_FAST_D256_CLK (referred as FOSC_DIV or 8m_d256/8md256 in TRM and reg. description) as RTC_SLOW_CLK source

enumerator **SOC_RTC_SLOW_CLK_SRC_INVALID**
Invalid RTC_SLOW_CLK source

enum **soc_rtc_fast_clk_src_t**
RTC_FAST_CLK mux inputs, which are the supported clock sources for the RTC_FAST_CLK.

**备注:** Enum values are matched with the register field values on purpose

**Values:**

enumerator **SOC_RTC_FAST_CLK_SRC_XTAL_D4**
Select XTAL_D4_CLK (may referred as XTAL_CLK_DIV_4) as RTC_FAST_CLK source

enumerator **SOC_RTC_FAST_CLK_SRC_XTAL_DIV**
Alias name for **SOC_RTC_FAST_CLK_SRC_XTAL_D4**

enumerator **SOC_RTC_FAST_CLK_SRC_RC_FAST**
Select RC_FAST_CLK as RTC_FAST_CLK source

enumerator **SOC_RTC_FAST_CLK_SRC_INVALID**
Invalid RTC_FAST_CLK source

enum **soc_module_clk_t**
Supported clock sources for modules (CPU, peripherals, RTC, etc.)

**备注:** enum starts from 1, to save 0 for special purpose

**Values:**

enumerator **SOC_MOD_CLK_CPU**
CPU_CLK can be sourced from XTAL, PLL, RC_FAST, or APLL by configuring soc_cpu_clk_src_t

enumerator **SOC_MOD_CLK_RTC_FAST**
RTC_FAST_CLK can be sourced from XTAL_D4 or RC_FAST by configuring soc_rtc_fast_clk_src_t

enumerator **SOC_MOD_CLK_RTC_SLOW**
RTC_SLOW_CLK can be sourced from RC_SLOW, XTAL32K, or RC_FAST_D256 by configuring soc_rtc_slow_clk_src_t

enumerator **SOC_MOD_CLK_APB**
APB_CLK is highly dependent on the CPU_CLK source
Chapter 2. API 参考

enumerator SOC_MOD_CLK_PLL_D2
    PLL_D2_CLK is derived from PLL, it has a fixed divider of 2

enumerator SOC_MOD_CLK_PLL_F160M
    PLL_F160M_CLK is derived from PLL, and has a fixed frequency of 160MHz

enumerator SOC_MOD_CLK_XTAL32K
    XTAL32K_CLK comes from the external 32kHz crystal, passing a clock gating to the peripherals

enumerator SOC_MOD_CLK_RC_FAST
    RC_FAST_CLK comes from the internal 8MHz rc oscillator, passing a clock gating to the peripherals

enumerator SOC_MOD_CLK_RC_FAST_D256
    RC_FAST_D256_CLK comes from the internal 8MHz rc oscillator, divided by 256, and passing a clock gating to the peripherals

enumerator SOC_MOD_CLK_XTAL
    XTAL_CLK comes from the external crystal (2~40MHz)

enumerator SOC_MOD_CLK_REF_TICK
    REF_TICK is derived from APB, it has a fixed frequency of 1MHz even when APB frequency changes

enumerator SOC_MOD_CLK_APLL
    APLL is sourced from PLL, and its frequency is configurable through APLL configuration registers

enum soc_periph_systimer_clk_src_t
    Type of SYSTIMER clock source.
    Values:
    enumerator SYSTIMER_CLK_SRC_XTAL
        SYSTIMER source clock is XTAL
    enumerator SYSTIMER_CLK_SRC_DEFAULT
        SYSTIMER source clock default choice is XTAL

enum soc_periph_gptimer_clk_src_t
    Type of GPTimer clock source.
    Values:
    enumerator GPTIMER_CLK_SRC_APB
        Select APB as the source clock
    enumerator GPTIMER_CLK_SRC_DEFAULT
        Select APB as the default choice

enum soc_periph_tg_clk_src_legacy_t
    Type of Timer Group clock source, reserved for the legacy timer group driver.
    Values:
enumerator **TIMER_SRC_CLK_APB**
Timer group source clock is APB

denumator **TIMER_SRC_CLK_DEFAULT**
Timer group source clock default choice is APB

denum **soc_periph_lcd_clk_src_t**
Type of LCD clock source.
*Values:*

denumator **LCD_CLK_SRC_PLL160M**
Select PLL_D2 (default to 160MHz) as the source clock

denumator **LCD_CLK_SRC_APLL**
Select APLL as the source clock

denumator **LCD_CLK_SRC_XTAL**
Select XTAL as the source clock

denumator **LCD_CLK_SRC_DEFAULT**
Select PLL_D2 (default to 160MHz) as the default choice

denum **soc_periph_rmt_clk_src_t**
Type of RMT clock source.
*Values:*

denumator **RMT_CLK_SRC_APB**
Select APB as the source clock

denumator **RMT_CLK_SRC_REF_TICK**
Select REF_TICK as the source clock

denumator **RMT_CLK_SRC_DEFAULT**
Select APB as the default choice

denum **soc_periph_rmt_clk_src_legacy_t**
Type of RMT clock source, reserved for the legacy RMT driver.
*Values:*

denumator **RMT_BASECLK_APB**
RMT source clock is APB CLK

denumator **RMT_BASECLK_REF**
RMT source clock is REF_TICK

denumator **RMT_BASECLK_DEFAULT**
RMT source clock default choice is APB
enum `soc_periph_uart_clk_src_legacy_t`
Type of UART clock source, reserved for the legacy UART driver.

*Values:*

enumerator `UART_SCLK_APB`
UART source clock is APB CLK

enumerator `UART_SCLK_REF_TICK`
UART source clock is REF_TICK

enumerator `UART_SCLK_DEFAULT`
UART source clock default choice is APB

enum `soc_periph_mcpwm_timer_clk_src_t`
Type of MCPWM timer clock source.

*Values:*

enumerator `MCPWM_TIMER_CLK_SRC_PLL160M`
Select PLL_F160M as the source clock

enumerator `MCPWM_TIMER_CLK_SRC_DEFAULT`
Select PLL_F160M as the default clock choice

enum `soc_periph_mcpwm_capture_clk_src_t`
Type of MCPWM capture clock source.

*Values:*

enumerator `MCPWM_CAPTURE_CLK_SRC_APB`
Select APB as the source clock

enumerator `MCPWM_CAPTURE_CLK_SRC_DEFAULT`
Select APB as the default clock choice

enum `soc_periph_i2s_clk_src_t`
I2S clock source enum.

*Values:*

enumerator `I2S_CLK_SRC_DEFAULT`
Select PLL_D2 as the default source clock

enumerator `I2S_CLK_SRC_PLL_160M`
Select PLL_D2 as the source clock

enumerator `I2S_CLK_SRC_APLL`
Select APLL as the source clock
enum `soc_periph_i2c_clk_src_t`
Type of I2C clock source.
*Values:*

enumerator `I2C_CLK_SRC_APB`
enumerator `I2C_CLK_SRC_DEFAULT`

enum `soc_periph_sdm_clk_src_t`
Sigma Delta Modulator clock source.
*Values:*

enumerator `SDM_CLK_SRC_APB`
   Select APB as the source clock
enumerator `SDM_CLK_SRC_DEFAULT`
   Select APB as the default clock choice

enum `soc_periph_dac_digi_clk_src_t`
DAC digital controller clock source.
*Values:*

enumerator `DAC_DIGI_CLK_SRC_PLLD2`
   Select PLLD2 as the source clock
enumerator `DAC_DIGI_CLK_SRC_APLL`
   Select APLL as the source clock
enumerator `DAC_DIGI_CLK_SRC_DEFAULT`
   Select PLLD2 as the default source clock

enum `soc_periph_dac_cosine_clk_src_t`
DAC cosine wave generator clock source.
*Values:*

enumerator `DAC_COSINE_CLK_SRC_RTC_FAST`
   Select RTC FAST as the source clock
enumerator `DAC_COSINE_CLK_SRC_DEFAULT`
   Select RTC FAST as the default source clock

enum `soc_periph_twai_clk_src_t`
TWAI clock source.
*Values:*

enumerator `TWAI_CLK_SRC_APB`
   Select APB as the source clock
### Digital To Analog Converter (DAC)

#### Overview

ESP32 has two 8-bit DAC (digital to analog converter) channels, connected to GPIO25 (Channel 1) and GPIO26 (Channel 2). Which means each channel of DAC can convert digital value 0~255 to the analog voltage 0~Vref, the output voltage can be calculate by:

\[
\text{out\_voltage} = \text{Vref} \times \frac{\text{digi\_val}}{255}
\]

The DAC peripheral supports outputting analog signal in following ways:

1. Outputting a voltage directly. The DAC channel will keep outputting a specified voltage.
2. Outputting continuous analog signal by DMA. The DAC will convert the data in a buffer at the specified frequency.
3. Outputting a cosine wave by the cosine wave generator. The DAC channel can output a cosine wave with specified frequency and amplitude.

For other analog output options, see the **Sigma-delta Modulation module** and the **LED Control module**. Both modules produce high-frequency PWM/PDM output, which can be hardware low-pass filtered in order to generate a lower frequency analog output.

#### DAC File Structure

Public headers that need to be included in the DAC application:

- **dac.h**: The top header file of legacy DAC driver, only included in the apps which use legacy driver API
- **dac_oneshot.h**: The top header file of new DAC driver, should be included in the apps which use the new driver API with oneshot mode.
- **dac_cosine.h**: The top header file of new DAC driver, should be included in the apps which use the new driver API with cosine mode.
- **dac_continuous.h**: The top header file of new DAC driver, should be included in the apps which use the new driver API with continuous mode.
Chapter 2. API

备注： The legacy driver can’t coexist with the new driver. Including dac.h to use the legacy driver or dac_oneshot.h, dac_cosine.h and dac_continuous.h to use the new driver. The legacy driver might be removed in future.

Functional Overview

Resources Management The DAC on ESP32 has two channels, due to the software resources are separate, they could be managed by the dac_oneshot_handle_t, dac_cosine_handle_t and dac_continuous_handle_t according to the usage. Of course, registering different modes on a same DAC channel is not allowed.

Direct Voltage Output (One-shot/Direct Mode) The DAC channels in the group can convert a 8-bit digital value into the analog every time calling dac_oneshot_output_voltage() (it can be called in ISR), and then the analog voltage will be kept on the DAC channel until next conversion start. To start to convert the voltage, the DAC channels need to be registered by dac_oneshot_new_channel() first, and the channel will be enabled after it is registered.

Continuous Wave Output (Continuous/DMA Mode) DAC channels can convert digital data continuously via the DMA. There are three ways to writing DAC data:

1. Normal writing (synchronous): It can transmit the data one time and keep blocking until all data has been loaded into the DMA buffer, and the voltage will be kept according to the last conversion value while no more data inputted, usually it is used to transport a long signal like an audio. To convert data continuously, the continuous channel handle need to be allocated by calling dac_continuous_new_channels() and the DMA conversion should be enabled by dac_continuous_enable(), then data can be written by dac_continuous_write() synchronously. Referring to peripherals/dac/dac_continuous/dac_audio for example.

2. Cyclical writing: It can convert a piece of data cyclically without blocking, no more operation needed after the data are loaded into the DMA buffer, but note that the inputted buffer size is limited by the descriptor number and the DMA buffer size, usually it is used to transport some short signal that need to be repeated, for example, a sine wave. To achieve cyclical writing, dac_continuous_write_cyclically() can be called after the DAC continuous mode is enabled. For the cyclical writing example, please refer to peripherals/dac/dac_continuous/signal_generator.

3. Asynchronous writing: It can transmit the data asynchronously based on the event callback. Thus dac_event_callbacks_t::on_convert_done must be registered to use asynchronous mode, and then users can get the dac_event_data_t in the callback which contains the DMA buffer address and length, allowing user to load the data into it directly. As mentioned, to use the asynchronous writing, dac_continuous_register_event_callback() need to be called to register the dac_event_callbacks_t::on_convert_done before enabling, and then calling dac_continuous_start_async_writing() to start the asynchronous writing, note that once the asynchronous writing started, the callback function will be triggered continuously, dac_continuous_write_asynchronously() can help to load the data either in a separate task or the callback directly. For the asynchronous example, please refer to peripherals/dac/dac_continuous/dac_audio as well.

On ESP32, DAC digital controller can be connected internally to the I2S0 and use its DMA for continuous conversion. Although the DAC only needs 8-bit data for conversion, it has to be left shifted 8 bits (i.e. the high 8 bits in 16-bit slot) to satisfy the I2S communication format. But the driver can help to expand automatically, if you want to expand manually, please disable CONFIG_DAC_DMA_AUTO_16BIT_ALIGN in the menuconfig.

The clock of DAC digital controller comes from I2S0 as well, so there are two kinds of clock source can be selected:

• dac_continuous_digi_clk_src_t::DAC_DIGI_CLK_SRC_PLL_D2 can support frequency between 19.6 KHz to several MHz. It is the default clock which can also be selected by dac_continuous_digi_clk_src_t::DAC_DIGI_CLK_SRC_DEFAULT.
• dac_continuous_dig_clk_src_t::DAC_DIGI_CLK_SRC_APLL can support frequency between 648 Hz to several MHz, however, it might be occupied by other peripherals, then it may not provide the required frequency. But it doesn’t mean APLL is not available in this case, it can still work as long as it can be divided to the target DAC DMA frequency correctly.

**Cosine Wave Output (Cosine Mode)** The DAC peripheral has one cosine wave generator, it can generate cosine wave on the channels, users can specify the frequency, amplitude and phase of the cosine wave. To output the cosine wave, please acquire the DAC to cosine mode handle by `dac_cosine_new_channel()` first, and then start the cosine wave generator by `dac_cosine_start()`.

Currently, the source clock of the cosine wave generator only comes from RTC_FAST which can be chosen by `dac_cosine_clk_src_t::DAC_COSINE_CLK_SRC_RTC_FAST`, it is also the default clock source which is same as `dac_cosine_clk_src_t::DAC_COSINE_CLK_SRC_RTC_DEFAULT`.

**Power Management** When the power management is enabled (i.e. `CONFIG_PM_ENABLE` is on), the system will adjust or stop the source clock of DAC before going into light sleep, thus potentially influence to the DAC signals may lead the data conversion goes wrong.

When using DAC driver in continuous mode, it can prevent the system from changing or stopping the source clock in DMA or cosine wave mode by acquiring a power management lock. When the source clock is generated from APB, the lock type will be set to `esp_pm_lock_type_t::ESP_PM_APB_FREQ_MAX` and when the source clock is APLL (only in DMA mode), it will be set to `esp_pm_lock_type_t::ESP_PM_NO_LIGHT_SLEEP`. Whenever the DAC is converting (i.e. DMA or cosine wave generator is working), the driver will guarantee that the power management lock is acquired after calling `dac_continuous_enable()`. Likewise, the driver will release the lock when `dac_continuous_disable()` is called.

**IRAM Safe** By default, the DAC DMA interrupt will be deferred when the Cache is disabled for reasons like writing/erasing Flash. Thus the DMA EOF interrupt will not get executed in time, which is not expected in a real-time application.

There’s a Kconfig option `CONFIG_DAC_ISR_IRAM_SAFE` that will:

1. Enable the interrupt being serviced even when cache is disabled
2. Place driver object into DRAM (in case it’s linked to PSRAM by accident)

This will allow the interrupt to run while the cache is disabled but will come at the cost of increased IRAM consumption.

**Thread Safety** All the public DAC APIs are guaranteed to be thread safe by the driver, which means, users can call them from different RTOS tasks without protection by extra locks. Notice that DAC driver uses mutex lock to ensure the thread safety, thus the APIs except `dac_oneshot_output_voltage()` are not allowed to be used in ISR.

**Kconfig Options**

- `CONFIG_DAC_ISR_IRAM_SAFE` controls whether the default ISR handler can work when cache is disabled, see IRAM Safe for more information.
- `CONFIG_DAC_SUPPRESS_DEPRECATED_WARN` controls whether to suppress the compiling warning message while using the legacy DAC driver.
- `CONFIG_DAC_ENABLE_DEBUG_LOG` is used to enabled the debug log output. Enable this option will increase the firmware binary size.
- `CONFIG_DAC_DMA_AUTO_16BIT_ALIGN` will auto expand the 8-bit data to 16-bit data in the driver to satisfy the I2S DMA format.
Chapter 2. API 参考

Application Example

The basic examples for the One-shot Mode, Continuous Mode and Cosine Mode can be found in:

- peripherals/dac/dac_oneshot
- peripherals/dac/dac_continuous
- peripherals/dac/dac_cosine_wave

API Reference

Header File

- components/driver/include/driver/dac_oneshot.h

Functions

```c
esp_err_t dac_oneshot_new_channel(const dac_oneshot_config_t *oneshot_cfg, dac_oneshot_handle_t *ret_handle)
```

Allocate a new DAC oneshot channel.

**备注**: The channel will be enabled as well when the channel allocated

参数

- oneshot_cfg - [in] The configuration for the oneshot channel
- ret_handle - [out] The returned oneshot channel handle

返回

- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The DAC channel has been registered already
- ESP_ERR_NO_MEM No memory for the DAC oneshot channel resources
- ESP_OK Allocate the new DAC oneshot channel success

```c
esp_err_t dac_oneshot_del_channel(dac_oneshot_handle_t handle)
```

Delete the DAC oneshot channel.

**备注**: The channel will be disabled as well when the channel deleted

参数

- handle - [in] The DAC oneshot channel handle

返回

- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The channel has already been de-registered
- ESP_OK Delete the oneshot channel success

```c
esp_err_t dac_oneshot_output_voltage(dac_oneshot_handle_t handle, uint8_t diigi_value)
```

Output the voltage.

**备注**: Generally it’ll take 7~11 us on ESP32 and 10~21 us on ESP32-S2

参数

- handle - [in] The DAC oneshot channel handle
- diigi_value  - [in] The digital value that need to be converted

返回

- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_OK Convert the digital value success
**Structures**

```c
struct dac_oneshot_config_t
    DAC oneshot channel configuration.
```

**Public Members**

```c
dac_channel_t chan_id
    DAC channel id
```

**Type Definitions**

```c
typedef struct dac_oneshot_s *dac_oneshot_handle_t
    DAConeshotchannelhandle
```

**Header File**

```c
• components/driver/include/driver/dac_cosine.h
```

**Functions**

```c
esp_err_t dac_cosine_new_channel (const dac_cosine_config_t *cos_cfg, dac_cosine_handle_t *ret_handle)
    Allocate a new DAC cosine wave channel.
```

**备注:** Since there is only one cosine wave generator, only the first channel can set the frequency of the cosine wave. Normally, the latter one is not allowed to set a different frequency, but the it can be forced to set by setting the bit force_set_freq in the configuration, notice that another channel will be affected as well when the frequency is updated.

```c
esp_err_t dac_cosine_del_channel (dac_cosine_handle_t handle)
    Delete the DAC cosine wave channel.
```

```c
esp_err_t dac_cosine_start (dac_cosine_handle_t handle)
    Start outputting the cosine wave on the channel.
```

**参数**

- **cos_cfg** [in] The configuration of cosine wave channel
- **ret_handle** [out] The returned cosine wave channel handle

**返回**

- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The DAC channel has been registered already
- ESP_ERR_NO_MEM No memory for the DAC cosine wave channel resources
- ESP_OK Allocate the new DAC cosine wave channel success

**参数**

- **handle** [in] The DAC cosine wave channel handle

**返回**

- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_OK OK The channel has already been deregistered
- ESP_OK Delete the cosine wave channel success

**参数**

- **handle** [in] The DAC cosine wave channel handle

**返回**

- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The channel has been started already
Chapter 2. API 参考

- ESP_OK Start the cosine wave success

```c
esp_err_t dac_cosine_stop(dac_cosine_handle_t handle)
```

Stop outputting the cosine wave on the channel.

参数 handle -[in] The DAC cosine wave channel handle

返回
- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERRINVALID_STATE The channel has been stopped already
- ESP_OK Stop the cosine wave success

**Structures**

struct dac_cosine_config_t

DAC cosine channel configurations.

**Public Members**

`dac_channel_t chan_id`

The cosine wave channel id

`uint32_t freq_hz`

The frequency of cosine wave, unit: Hz. The cosine wave generator is driven by RTC_FAST clock which is divide from RC_FAST. With the default RTC clock, the minimum frequency of cosine wave is about 130 Hz. Although it can support up to several MHz frequency theoretically, the waveform will distort at high frequency due to the hardware limitation. Typically not suggest to set the frequency higher than 200 KHz

`dac_cosine_clk_src_t clk_src`

The clock source of the cosine wave generator, currently only support

DAC_COSINE_CLK_SRC_DEFAULT

`dac_cosine_atten_t atten`

The attenuation of cosine wave amplitude

`dac_cosine_phase_t phase`

The phase of cosine wave, can only support DAC_COSINE_PHASE_0 or DAC_COSINE_PHASE_180, default as 0 while setting an unsupported phase

`int8_t offset`

The DC offset of cosine wave

`bool force_set_freq`

Force to set the cosine wave frequency

struct dac_cosine_config_t:[anonymous] flags

Flags of cosine mode

**Type Definitions**

typedef struct dac_cosine_s *dac_cosine_handle_t

DAC cosine wave channel handle
Chapter 2. API 参考

Header File

- components/driver/include/driver/dac_continuous.h

Functions

esp_err_t dac_continuous_new_channels(const dac_continuous_config_t *cont_cfg, dac_continuous_handle_t *ret_handle)

Allocate new DAC channels in continuous mode.

备注：The DAC channels can’t be registered to continuous mode separately

参数
- cont_cfg [in] Continuous mode configuration
- ret_handle [out] The returned continuous mode handle

返回
- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The DAC channel has been registered already
- ESP_ERR_NOT_FOUND Not found the available dma peripheral, might be occupied
- ESP_ERR_NO_MEM No memory for the DAC continuous mode resources
- ESP_OK Allocate the new DAC continuous mode success

esp_err_t dac_continuous_del_channels(dac_continuous_handle_t handle)

Delete the DAC continuous handle.

参数 handle *[in] The DAC continuous channel handle that obtained from 'dac_continuous_new_channels'

返回
- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The channels have already been deregistered or not disabled
- ESP_OK Delete the continuous channels success

esp_err_t dac_continuous_enable(dac_continuous_handle_t handle)

Enabled the DAC continuous mode.

备注：Must enable the channels before

参数 handle *[in] The DAC continuous channel handle that obtained from 'dac_continuous_new_channels'

返回
- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The channels have been enabled already
- ESP_OK Enable the continuous output success

esp_err_t dac_continuous_disable(dac_continuous_handle_t handle)

Disable the DAC continuous mode.

参数 handle *[in] The DAC continuous channel handle that obtained from 'dac_continuous_new_channels'

返回
- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The channels have been enabled already
- ESP_OK Disable the continuous output success
**esp_err_t** `dac_continuous_write` *(dac_continuous_handle_t handle, uint8_t* buf, size_t buf_size, size_t *bytes_loaded, int timeout_ms)*

Write DAC data continuously.

**备注:** The data in buffer will only be converted one time. This function will be blocked until all data loaded or timeout then the DAC output will keep outputting the voltage of the last data in the buffer.

**参数:**
- **handle** –[in] The DAC continuous channel handle that obtained from `dac_continuous_new_channels`
- **buf** –[in] The digital data buffer to convert
- **buf_size** –[in] The buffer size of digital data buffer
- **bytes_loaded** –[out] The bytes that has been loaded into DMA buffer, can be NULL if don’t need it
- **timeout_ms** –[in] The timeout time in millisecond, set a minus value means will block forever

**返回:**
- ESP_ERR_INVALID_ARG The input parameter is invalid
- ESP_ERR_INVALID_STATE The DAC continuous mode has not been enabled yet
- ESP_ERR_TIMEOUT Waiting for semaphore or message queue timeout
- ESP_OK Success to output the acyclic DAC data

**esp_err_t** `dac_continuous_write_cyclically` *(dac_continuous_handle_t handle, uint8_t* buf, size_t buf_size, size_t *bytes_loaded)*

Write DAC continuous data cyclically.

**备注:** The data in buffer will be converted cyclically using DMA once this function is called, This function will return once the data loaded into DMA buffers.

**备注:** The buffer size of cyclically output is limited by the descriptor number and dma buffer size while initializing the continuous mode. Concretely, in order to load all the data into descriptors, the cyclic buffer size is not supposed to be greater than `desc_num * buf_size`

**备注:** Specially, on ESP32, the data bit width of DAC continuous data is fixed to 16 bits while only the high 8 bits are available. The driver will help to expand the inputted buffer automatically by default, you can also align the data to 16 bits manually by clearing `CONFIG_DAC_DMA_AUTO_16BIT_ALIGN` in menuconfig.

**参数:**
- **handle** –[in] The DAC continuous channel handle that obtained from `dac_continuous_new_channels`
- **buf** –[in] The digital data buffer to convert
- **buf_size** –[in] The buffer size of digital data buffer
- **bytes_loaded** –[out] The bytes that has been loaded into DMA buffer, can be NULL if don’t need it

**返回:**
- ESP_ERR_INVALID_ARG The input parameter is invalid
Chapter 2. API

- ESP_ERR_INVALID_STATE The DAC continuous mode has not been enabled yet
- ESP_OK Success to output the acyclic DAC data

\[ \text{esp_err_t dac_continuous_register_event_callback (dac_continuous_handle_t handle, const dac_event_callbacks_t *callbacks, void *user_data)} \]

Set event callbacks for DAC continuous mode.

**备注**: User can deregister a previously registered callback by calling this function and setting the callback member in the `callbacks` structure to NULL.

**备注**: When CONFIG_DAC_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in this function, including the `user_data`, should be in the internal RAM as well.

参数
- `handle` – [in] The DAC continuous channel handle that obtained from `dac_continuous_new_channels`
- `callbacks` – [in] Group of callback functions, input NULL to clear the former callbacks
- `user_data` – [in] User data, which will be passed to callback functions directly

返回
- ESP_OK Set event callbacks successfully
- ESP_ERR_INVALID_ARG Set event callbacks failed because of invalid argument

\[ \text{esp_err_t dac_continuous_start_async_writing (dac_continuous_handle_t handle)} \]

Start the async writing.

**备注**: When the asynchronous writing start, the DAC will keep outputting ‘0’ until the data are loaded into the DMA buffer. To loaded the data into DMA buffer, ‘on_convert_done’ callback is required, which can be registered by `dac_continuous_register_event_callback` before enabling

参数 `handle` – [in] The DAC continuous channel handle that obtained from `dac_continuous_new_channels`

返回
- ESP_OK Start asynchronous writing successfully
- ESP_ERR_INVALID_ARG The handle is NULL
- ESP_ERR_INVALID_STATE The channel is not enabled or the ‘on_convert_done’ callback is not registered

\[ \text{esp_err_t dac_continuous_stop_async_writing (dac_continuous_handle_t handle)} \]

Stop the sync writing.

参数 `handle` – [in] The DAC continuous channel handle that obtained from `dac_continuous_new_channels`

返回
- ESP_OK Stop asynchronous writing successfully
- ESP_ERR_INVALID_ARG The handle is NULL
- ESP_ERR_INVALID_STATE Asynchronous writing has not started

\[ \text{esp_err_t dac_continuous_write_asynchronously (dac_continuous_handle_t handle, uint8_t dma_buf, size_t dma_buf_len, const uint8_t *data, size_t data_len, size_t *bytes_loaded)} \]

Write DAC data asynchronously.
This function can be called when the asynchronous writing started, and it can be called in the callback directly but recommend to writing data in a task, referring to :example:`peripherals/dac/dac_continuous/dac_audio`

参数

- **handle** [-in] The DAC continuous channel handle that obtained from `dac_continuous_new_channels`
- **dma_buf** [-in] The DMA buffer address, it can be acquired from `dac_event_data_t` in the `on_convert_done` callback
- **dma_buf_len** [-in] The DMA buffer length, it can be acquired from `dac_event_data_t` in the `on_convert_done` callback
- **data** [-in] The data that need to be written
- **data_len** [-in] The data length the need to be written
- **bytes_loaded** [-out] The bytes number that has been loaded/written into the DMA buffer

返回

- ESP_OK Write the data into DMA buffer successfully
- ESP_ERR_INVALID_ARG NULL pointer
- ESP_ERR_INVALID_STATE The channels haven’t start the asynchronous writing
- ESP_ERR_NOT_FOUND The param ‘dam_buf’ not match any existed DMA buffer

Structures

struct **dac_continuous_config_t**

DAC continuous channels’ configurations.

Public Members

`dac_channel_mask_t chan_mask`

DAC channels’ mask for selecting which channels are used

`uint32_t desc_num`

The number of DMA descriptor, at least 2 descriptors are required The number of descriptors is directly proportional to the max data buffer size while converting in cyclic output but only need to ensure it is greater than 1 in acyclic output Typically, suggest to set the number bigger than 5, in case the DMA stopped while sending a short buffer

`size_t buf_size`

The DMA buffer size, should be within 32~4092 bytes. Each DMA buffer will be attached to a DMA descriptor, i.e., the number of DMA buffer will be equal to the DMA descriptor number The DMA buffer size is not allowed to be greater than 4092 bytes The total DMA buffer size equal to desc_num * buf_size Typically, suggest to set the size to the multiple of 4

`uint32_t freq_hz`

The frequency of DAC conversion in continuous mode, unit: Hz The supported range is related to the target and the clock source. For the clock `DAC_DIGI_CLK_SRC_DEFAULT`: the range is 19.6 KHz to several MHz on ESP32 and 77 Hz to several MHz on ESP32-S2. For the clock `DAC_DIGI_CLK_SRC_APLL`: the range is 648 Hz to several MHz on ESP32 and 6 Hz to several MHz on ESP32-S2. Typically not suggest to set the frequency higher than 2 MHz, otherwise the severe distortion will appear
### Chapter 2. API 参考

#### int8_t offset

The offset of the DAC digital data. Range -128~127

#### dac_continuous_digi_clk_src_t clk_src

The clock source of digital controller, which can affect the range of supported frequency. Currently DAC_DIGI_CLK_SRC_DEFAULT and DAC_DIGI_CLK_SRC_APLL are available.

#### dac_continuous_channel_mode_t chan_mode

The channel mode of continuous mode, only take effect when multiple channels enabled, depends converting the buffer alternately or simultaneously.

#### struct dac_event_data_t

Event structure used in DAC event queue.

### Public Members

#### void *buf

The pointer of DMA buffer that just finished sending.

#### size_t buf_size

The writable buffer size of the DMA buffer, equal to `dac_continuous_config_t::buf_size`

#### size_t write_bytes

The number of bytes that be written successfully.

#### struct dac_event_callbacks_t

Group of DAC callbacks.

<table>
<thead>
<tr>
<th>备注</th>
<th>The callbacks are all running under ISR environment</th>
</tr>
</thead>
</table>

| 备注 | When CONFIG_DAC_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. |

### Public Members

#### dac_isr_callback_t on_convert_done

Callback of data conversion done event. An event data buffer previously loaded to the driver has been output and converted. The event data includes DMA buffer address and size that just finished converting.

#### dac_isr_callback_t on_stop

Callback of finished sending all the data. All loaded event data buffers are converted. Driver is pending for new data buffers to be loaded. The event data will be NULL in this callback.

### Type Definitions

typedef struct dac_continuous_s *dac_continuous_handle_t

DAC continuous channel handle
typedef bool (*dac_isr_callback_t)(dac_continuous_handle_t *handle, const dac_event_data_t *event, void *user_data)

DAC event callback.

**Param handle** [in] DAC channel handle, created from dac_continuous_new_channels()

**Param event** [in] DAC event data

**Param user_data** [in] User registered context, passed from dac_continuous_register_event_callback()

**Return** Whether a high priority task has been waken up by this callback function

### Enumerations

**enum dac_channel_mask_t**

DAC channel mask.

**Values:**

- enumerator **DAC_CHANNEL_MASK_CH0**
  
  DAC channel 0 is GPIO25(ESP32) / GPIO17(ESP32S2)

- enumerator **DAC_CHANNEL_MASK_CH1**
  
  DAC channel 1 is GPIO26(ESP32) / GPIO18(ESP32S2)

- enumerator **DAC_CHANNEL_MASK_ALL**
  
  Both DAC channel 0 and channel 1

---

### Header File

- components/driver/include/driver/dac_types.h

---

### Type Definitions

**typedef soc_periph_dac_digi_clk_src_t dac_continuous_digi_clk_src_t**

DAC DMA (digital controller) clock source.

**typedef soc_periph_dac_cosine_clk_src_t dac_cosine_clk_src_t**

DAC cosine wave generator clock source.

---

### Enumerations

**enum dac_continuous_channel_mode_t**

DAC channel work mode in dma mode.

---

**备注：** Only take effect when multiple channels enabled.

---

**备注：** Assume the data in buffer is ‘A B C D E F’

- DAC_CHANNEL_MODE_SIMUL:
  - channel 0: A B C D E F
  - channel 1: A B C D E F

- DAC_CHANNEL_MODE_ALTER:
  - channel 0: A C E
  - channel 1: B D F
Chapter 2. API

Values:

enumerator `DAC_CHANNEL_MODE_SIMUL`
   The data in the DMA buffer is simultaneously output to the enable channel of the DAC.

enumerator `DAC_CHANNEL_MODE_ALTER`
   The data in the DMA buffer is alternately output to the enable channel of the DAC.

Header File

- components/hal/include/hal/dac_types.h

Enumerations

enum `dac_channel_t`

Values:

enumerator `DAC_CHAN_0`
   DAC channel 0 is GPIO25(ESP32) / GPIO17(ESP32S2)

enumerator `DAC_CHAN_1`
   DAC channel 1 is GPIO26(ESP32) / GPIO18(ESP32S2)

enumerator `DAC_CHANNEL_1`
   Alias of ‘DAC_CHAN_0’, now the channel index start from ‘0’

enumerator `DAC_CHANNEL_2`
   Alias of ‘DAC_CHAN_1’, now the channel index start from ‘0’

enum `dac_cosine_atten_t`

The attenuation of the amplitude of the cosine wave generator. The max amplitude is VDD3P3_RTC.

Values:

enumerator `DAC_COSINE_ATTEN_DEFAULT`
   No attenuation to the DAC cosine wave amplitude. Default.

enumerator `DAC_COSINE_ATTEN_DB_0`
   Original amplitude of the DAC cosine wave, equals to DAC_COSINE_ATTEN_DEFAULT

enumerator `DAC_COSINE_ATTEN_DB_6`
   1/2 amplitude of the DAC cosine wave

enumerator `DAC_COSINE_ATTEN_DB_12`
   1/4 amplitude of the DAC cosine wave

enumerator `DAC_COSINE_ATTEN_DB_18`
   1/8 amplitude of the DAC cosine wave
enum **dac_cosine_phase_t**

Set the phase of the cosine wave generator output.

**Values:**

enumerator **DAC_COSINE_PHASE_0**
Phase shift +0°

enumerator **DAC_COSINE_PHASE_180**
Phase shift +180°

### 2.6.6 GPIO & RTC GPIO

**GPIO** 汇总

ESP32芯片具有34个物理GPIO管脚（GPIO0 ~ GPIO19，GPIO21 ~ GPIO23，GPIO25 ~ GPIO27和GPIO32 ~ GPIO39）。每个管脚都可用作一个通用IO，或连接一个内部的外设信号。通过IO MUX、RTC IO MUX和GPIO交换矩阵，可配置外设模块的输入信号来源于任何的IO管脚，并且外设模块的输出信号也可连接到任意IO管脚。这些模块共同组成了芯片的IO控制。更多详细信息，请参阅ESP32技术参考手册> IO MUX和GPIO矩阵（GPIO，IO_MUX）[PDF]。

下表提供了各管脚的详细信息，部分GPIO具有特殊的使用限制，具体可参考表中的注释列。

<table>
<thead>
<tr>
<th>GPIO</th>
<th>模拟功能</th>
<th>RTC GPIO</th>
<th>注释</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO0</td>
<td>ADC2.CH1</td>
<td>RTC_GPIO11</td>
<td>Strapping 管脚</td>
</tr>
<tr>
<td>GPIO1</td>
<td></td>
<td>TXD</td>
<td></td>
</tr>
<tr>
<td>GPIO2</td>
<td>ADC2.CH2</td>
<td>RTC_GPIO12</td>
<td>Strapping 管脚</td>
</tr>
<tr>
<td>GPIO3</td>
<td>ADC2.CH0</td>
<td>RTC_GPIO10</td>
<td></td>
</tr>
<tr>
<td>GPIO5</td>
<td></td>
<td></td>
<td>Strapping 管脚</td>
</tr>
<tr>
<td>GPIO6</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO7</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO8</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO9</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO10</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO11</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO12</td>
<td>ADC2.CH5</td>
<td>RTC_GPIO15</td>
<td>Strapping 管脚；JTAG</td>
</tr>
<tr>
<td>GPIO13</td>
<td>ADC2.CH4</td>
<td>RTC_GPIO14</td>
<td>JTAG</td>
</tr>
<tr>
<td>GPIO14</td>
<td>ADC2.CH6</td>
<td>RTC_GPIO16</td>
<td>JTAG</td>
</tr>
<tr>
<td>GPIO15</td>
<td>ADC2.CH3</td>
<td>RTC_GPIO13</td>
<td>Strapping 管脚；JTAG</td>
</tr>
<tr>
<td>GPIO16</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO17</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO18</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO19</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO21</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO22</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO23</td>
<td></td>
<td>SPI0/1</td>
<td></td>
</tr>
<tr>
<td>GPIO25</td>
<td>ADC2.CH8</td>
<td>RTC_GPIO6</td>
<td></td>
</tr>
</tbody>
</table>
表 2 - 续上页

<table>
<thead>
<tr>
<th>GPIO</th>
<th>模拟功能</th>
<th>RTC GPIO</th>
<th>注释</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO26</td>
<td>ADC2_CH9</td>
<td>RTC_GPIO7</td>
<td></td>
</tr>
<tr>
<td>GPIO27</td>
<td>ADC2_CH7</td>
<td>RTC_GPIO17</td>
<td></td>
</tr>
<tr>
<td>GPIO32</td>
<td>ADC1_CH4</td>
<td>RTC_GPIO9</td>
<td></td>
</tr>
<tr>
<td>GPIO33</td>
<td>ADC1_CH5</td>
<td>RTC_GPIO8</td>
<td></td>
</tr>
<tr>
<td>GPIO34</td>
<td>ADC1_CH6</td>
<td>RTC_GPIO4</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIO35</td>
<td>ADC1_CH7</td>
<td>RTC_GPIO5</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIO36</td>
<td>ADC1_CH0</td>
<td>RTC_GPIO0</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIO37</td>
<td>ADC1_CH1</td>
<td>RTC_GPIO1</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIO38</td>
<td>ADC1_CH2</td>
<td>RTC_GPIO2</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIO39</td>
<td>ADC1_CH3</td>
<td>RTC_GPIO3</td>
<td>GPIO</td>
</tr>
</tbody>
</table>

备注：
• Strapping 管脚：GPIO0, GPIO2, GPIO5, GPIO12 (MTDI) 和 GPIO15 (MTDO) 是 Strapping 管脚。更多信息请参考 ESP32 技术规格书。
• SPI0/1：GPIO6-11 和 GPIO16-17 通常连接到模块内的 SPI flash 和 PSRAM，因此不能用于其他用途。
• JTAG：GPIO12-15 通常用于在线调试。
• GPIO：GPIO34-39 只能设置为输入模式，不具备软件使能的上拉或下拉功能。
• TXD & RXD 通常用于烧录和调试。
• ADC2：使用 Wi-Fi 时不能使用 ADC2 管脚。因此，如果您在使用 Wi-Fi 时无法从 ADC2 GPIO 获取值，可以考虑使用 ADC1 GPIO 来解决该问题。更多详情请参考 ADC 连续转换模式下的硬件限制以及 ADC 单次转换模式下的硬件限制。
• 使用 ADC 或睡眠模式下使用 Wi-Fi 和蓝牙时，请不要使用 GPIO36 和 GPIO39 的中断。有关问题的详细描述，请参考 ESP32 ECO 和 Bug 解决方法 > 中的第 3.11 节。

当 GPIO 连接到“RTC”低功耗模拟子系统时，ESP32 芯片还单独支持“RTC GPIO”。可在以下情况时使用这些管脚功能：

- 处于 Deep-sleep 模式时
- 超低功耗处理器 (ULP) 运行时
- 使用 ADC/DAC 等模拟功能时

应用示例

- GPIO 输出和输入中断示例：peripherals/gpio/generic_gpio。

API 参考 - 普通 GPIO

Header File

- components/driver/include/driver/gpio.h

Functions

```c
esp_err_t gpio_config(const gpio_config_t *pGPIOConfig)
```

GPIO common configuration.

参数 pGPIOConfig - Pointer to GPIO configure struct
**Chapter 2. API 参考**

**API**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpio_reset_pin(gpio_num_t gpio_num)</code></td>
<td>Reset an gpio to default state (select gpio function, enable pullup and disable input and output).</td>
</tr>
</tbody>
</table>

**Note:** This function also configures the IOMUX for this pin to the GPIO function, and disconnects any other peripheral output configured via GPIO Matrix.

**GPIO number.**

**Function**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpio_set_intr_type(gpio_num_t gpio_num, gpio_int_type_t intr_type)</code></td>
<td>GPIO set interrupt trigger type.</td>
</tr>
</tbody>
</table>

**Parameters**

- `gpio_num` - GPIO number. If you want to set the trigger type of e.g. of GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);
- `intr_type` - Interrupt type, select from `gpio_int_type_t`;

**Return**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpio_intr_enable(gpio_num_t gpio_num)</code></td>
<td>Enable GPIO module interrupt signal.</td>
</tr>
</tbody>
</table>

**Note:** ESP32: Please do not use the interrupt of GPIO36 and GPIO39 when using ADC or Wi-Fi and Bluetooth with sleep mode enabled. Please refer to the comments of `adc1_get_raw`. Please refer to Section 3.11 of ESP32 ECO and Workarounds for Bugs for the description of this issue.

**GPIO number.**

**Function**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpio_intr_disable(gpio_num_t gpio_num)</code></td>
<td>Disable GPIO module interrupt signal.</td>
</tr>
</tbody>
</table>

**Note:** This function is allowed to be executed when Cache is disabled within ISR context, by enabling `CONFIG_GPIO_CTRL_FUNC_IN_IRAM`

**GPIO number.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpio_set_level(gpio_num_t gpio_num, uint32_t level)</code></td>
<td>GPIO set output level.</td>
</tr>
</tbody>
</table>

**Parameters**

- `gpio_num` - GPIO number. If you want to enable an interrupt on e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);

**Return**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**Function**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpio_set_intr_type(gpio_num_t gpio_num, gpio_int_type_t intr_type)</code></td>
<td>GPIO set interrupt trigger type.</td>
</tr>
</tbody>
</table>

**Parameters**

- `gpio_num` - GPIO number. If you want to disable the interrupt of e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);

**Return**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**Function**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpio_set_level(gpio_num_t gpio_num, uint32_t level)</code></td>
<td>GPIO set output level.</td>
</tr>
</tbody>
</table>
Chapter 2. API 参考

备注: This function is allowed to be executed when Cache is disabled within ISR context, by enabling CONFIG_GPIO_CTRL_FUNC_IN_IRAM

参数
- **gpio_num** - GPIO number. If you want to set the output level of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **level** - Output level. 0: low ; 1: high

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO number error

int **gpio_get_level**(gpio_num_t gpio_num)

GPIO get input level.

警告: If the pad is not configured for input (or input and output) the returned value is always 0.

参数 **gpio_num** - GPIO number. If you want to get the logic level of e.g. pin GPIO16, gpio_num should be GPIO_NUM_16 (16);

返回
- 0 the GPIO input level is 0
- 1 the GPIO input level is 1

**esp_err_t gpio_set_direction**(gpio_num_t gpio_num, gpio_mode_t mode)

GPIO set direction.

Configure GPIO direction, such as output_only, input_only, output_and_input

参数
- **gpio_num** - Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **mode** - GPIO direction

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO error

**esp_err_t gpio_set_pull_mode**(gpio_num_t gpio_num, gpio_pull_mode_t pull)

Configure GPIO pull-up/pull-down resistors.

备注: ESP32: Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.

参数
- **gpio_num** - GPIO number. If you want to set pull up or down mode for e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **pull** - GPIO pull up/down mode.

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG: Parameter error

**esp_err_t gpio_wakeup_enable**(gpio_num_t gpio_num, gpio_int_type_t intr_type)

Enable GPIO wake-up function.

参数
- **gpio_num** - GPIO number.
Chapter 2. API

- **intr_type** - GPIO wake-up type. Only GPIO_INTR_LOW_LEVEL or GPIO_INTR_HIGH_LEVEL can be used.

  **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG Parameter error

**esp_err_t gpio_wakeup_disable (gpio_num_t gpio_num)**

Disable GPIO wake-up function.

  **参数**
  - gpio_num - GPIO number

  **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG Parameter error

**esp_err_t gpio_isr_register (void (*fn)(void*), void*arg, int intr_alloc_flags, gpio_isr_handle_t*handle)**

Register GPIO interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

This ISR function is called whenever any GPIO interrupt occurs. See the alternative gpio_install_isr_service() and gpio_isr_handler_add() API in order to have the driver support per-GPIO ISRs.

To disable or remove the ISR, pass the returned handle to the interrupt allocation functions.

  **参数**
  - fn - Interrupt handler function.
  - arg - Parameter for handler function
  - intr_alloc_flags - Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_ *values. See esp_intr_alloc.h for more info.
  - handle - Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

  **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG GPIO error
  - ESP_ERR_NOT_FOUND No free interrupt found with the specified flags

**esp_err_t gpio_pullup_en (gpio_num_t gpio_num)**

Enable pull-up on GPIO.

  **参数**
  - gpio_num - GPIO number

  **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG Parameter error

**esp_err_t gpio_pullup_dis (gpio_num_t gpio_num)**

Disable pull-up on GPIO.

  **参数**
  - gpio_num - GPIO number

  **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG Parameter error

**esp_err_t gpio_pulldown_en (gpio_num_t gpio_num)**

Enable pull-down on GPIO.

  **参数**
  - gpio_num - GPIO number

  **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG Parameter error

**esp_err_t gpio_pulldown_dis (gpio_num_t gpio_num)**

Disable pull-down on GPIO.

  **参数**
  - gpio_num - GPIO number

  **返回**
Chapter 2. API

esp_err_t gpio_install_isr_service(int intr_alloc_flags)
Install the GPIO driver’s ETS_GPIO_INTR_SOURCE ISR handler service, which allows per-pin GPIO interrupt handlers.

This function is incompatible with gpio_isr_register() - if that function is used, a single global ISR is registered for all GPIO interrupts. If this function is used, the ISR service provides a global GPIO ISR and individual pin handlers are registered via the gpio_isr_handler_add() function.

参数 intr_alloc_flags – Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_*. * values. See esp_intr_alloc.h for more info.

返回
ESP_OK Success
ESP_ERR_NO_MEM No memory to install this service
ESP_ERR_INVALID_STATE ISR service already installed.
ESP_ERR_NOT_FOUND No free interrupt found with the specified flags
ESP_ERR_INVALID_ARG GPIO error

void gpio_uninstall_isr_service()
Uninstall the driver’s GPIO ISR service, freeing related resources.

esp_err_t gpio_isr_handler_add(gpio_num_t gpio_num, gpio_isr_t isr_handler, void *args)
Add ISR handler for the corresponding GPIO pin.

Call this function after using gpio_install_isr_service() to install the driver’s GPIO ISR handler service.

The pin ISR handlers no longer need to be declared with IRAM_ATTR, unless you pass the ESP_INTR_FLAG_IRAM flag when allocating the ISR in gpio_install_isr_service().

This ISR handler will be called from an ISR. So there is a stack size limit (configurable as “ISR stack size” in menuconfig). This limit is smaller compared to a global GPIO interrupt handler due to the additional level of indirection.

参数
- gpio_num – GPIO number
- isr_handler – ISR handler function for the corresponding GPIO number.
- args – parameter for ISR handler.

返回
ESP_OK Success
ESP_ERR_INVALID_STATE Wrong state, the ISR service has not been initialized.
ESP_ERR_INVALID_ARG Parameter error

esp_err_t gpio_isr_handler_remove(gpio_num_t gpio_num)
Remove ISR handler for the corresponding GPIO pin.

参数 gpio_num – GPIO number

返回
ESP_OK Success
ESP_ERR_INVALID_STATE Wrong state, the ISR service has not been initialized.
ESP_ERR_INVALID_ARG Parameter error

esp_err_t gpio_set_drive_capability(gpio_num_t gpio_num, gpio_drive_cap_t strength)
Set GPIO pad drive capability.

参数
- gpio_num – GPIO number, only support output GPIOs
- strength – Drive capability of the pad

返回
ESP_OK Success
ESP_ERR_INVALID_ARG Parameter error
Chapter 2. API

```c
esp_err_t gpio_get_drive_capability (gpio_num_t gpio_num, gpio_drive_cap_t *strength)
```

Get GPIO pad drive capability.

### Parameters
- **gpio_num** - GPIO number, only support output GPIOs
- **strength** - Pointer to accept drive capability of the pad

### Return Values
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

```c
esp_err_t gpio_hold_en (gpio_num_t gpio_num)
```

Enable gpio pad hold function.

When the pin is set to hold, the state is latched at that moment and will not change no matter how the internal signals change or how the IO MUX/GPIO configuration is modified (including input enable, output enable, output value, function, and drive strength values). It can be used to retain the pin state through a core reset and system reset triggered by watchdog time-out or Deep-sleep events.

The gpio pad hold function works in both input and output modes, but must be output-capable gpios. If pad hold enabled: in output mode: the output level of the pad will be force locked and can not be changed. in input mode: input read value can still reflect the changes of the input signal.

The state of the digital gpio cannot be held during Deep-sleep, and it will resume to hold at its default pin state when the chip wakes up from Deep-sleep. If the digital gpio also needs to be held during Deep-sleep, `gpio_deep_sleep_hold_en` should also be called.

Power down or call `gpio_hold_dis` will disable this function.

### Parameters
- **gpio_num** - GPIO number, only support output-capable GPIOs

### Return Values
- ESP_OK Success
- ESP_ERR_NOT_SUPPORTED Not support pad hold function

```c
esp_err_t gpio_hold_dis (gpio_num_t gpio_num)
```

Disable gpio pad hold function.

When the chip is woken up from Deep-sleep, the gpio will be set to the default mode, so, the gpio will output the default level if this function is called. If you don’t want the level changes, the gpio should be configured to a known state before this function is called. e.g. If you hold gpio18 high during Deep-sleep, after the chip is woken up and `gpio_hold_dis` is called, gpio18 will output low level(because gpio18 is input mode by default). If you don’t want this behavior, you should configure gpio18 as output mode and set it to high level before calling `gpio_hold_dis`.

### Parameters
- **gpio_num** - GPIO number, only support output-capable GPIOs

### Return Values
- ESP_OK Success
- ESP_ERR_NOT_SUPPORTED Not support pad hold function

```c
void gpio_deep_sleep_hold_en (void)
```

Enable all digital gpio pads hold function during Deep-sleep.

Enabling this feature makes all digital gpio pads be at the holding state during Deep-sleep. The state of each pad holds is its active configuration (not pad’s sleep configuration!).

Note that this pad hold feature only works when the chip is in Deep-sleep mode. When the chip is in active mode, the digital gpio state can be changed freely even you have called this function.

After this API is being called, the digital gpio Deep-sleep hold feature will work during every sleep process. You should call `gpio_deep_sleep_hold_dis` to disable this feature.

```c
void gpio_deep_sleep_hold_dis (void)
```

Disable all digital gpio pads hold function during Deep-sleep.
**void gpio_iomux_in (uint32_t gpio_num, uint32_t signal_idx)**

Set pad input to a peripheral signal through the IOMUX.

- **gpio_num** - GPIO number of the pad.
- **signal_idx** - Peripheral signal id to input. One of the *_IN_IDX signals in soc/gpio_sig_map.h.

**void gpio_iomux_out (uint8_t gpio_num, int func, bool oen_inv)**

Set peripheral output to an GPIO pad through the IOMUX.

- **gpio_num** - GPIO number of the pad.
- **func** - The function number of the peripheral pin to output pin. One of the FUNC_X_* of specified pin (X) in soc/io_mux_reg.h.
- **oen_inv** - True if the output enable needs to be inverted, otherwise False.

**esp_err_t gpio_sleep_sel_en (gpio_num_t gpio_num)**

Enable SLP_SEL to change GPIO status automatically in lightsleep.

- **gpio_num** - GPIO number of the pad.
  - **ESP_OK** Success

**esp_err_t gpio_sleep_sel_dis (gpio_num_t gpio_num)**

Disable SLP_SEL to change GPIO status automatically in lightsleep.

- **gpio_num** - GPIO number of the pad.
  - **ESP_OK** Success

**esp_err_t gpio_sleep_set_direction (gpio_num_t gpio_num, gpio_mode_t mode)**

GPIO set direction at sleep.

Configure GPIO direction, such as output_only, input_only, output_and_input

- **gpio_num** - Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
  - **mode** - GPIO direction
  - **ESP_OK** Success
  - **ESP_ERR_INVALID_ARG** GPIO error

**esp_err_t gpio_sleep_set_pull_mode (gpio_num_t gpio_num, gpio_pull_mode_t pull)**

Configure GPIO pull-up/pull-down resistors at sleep.

备注: ESP32: Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.
struct `gpio_config_t`  
Configuration parameters of GPIO pad for `gpio_config` function.

**Public Members**

`uint64_t pin_bit_mask`
GPIO pin: set with bitmask, each bit maps to a GPIO

`gpio_mode_t mode`
GPIO mode: set input/output mode

`gpio_pullup_t pull_up_en`
GPIO pull-up

`gpio_pulldown_t pull_down_en`
GPIO pull-down

`gpio_int_type_t intr_type`
GPIO interrupt type

**Macros**

`GPIO_PIN_COUNT`

`GPIO_IS_VALID_GPIO(gpio_num)`  
Check whether it is a valid GPIO number.

`GPIO_IS_VALID_OUTPUT_GPIO(gpio_num)`  
Check whether it can be a valid GPIO number of output mode.

`GPIO_IS_VALID_DIGITAL_IO_PAD(gpio_num)`  
Check whether it can be a valid digital I/O pad.

**Type Definitions**

typedef `intr_handle_t gpio_isr_handle_t`

typedef void (`*gpio_isr_t`(void *arg)`)
GPIO interrupt handler.

**Param arg**  
User registered data

**Header File**

- components/hal/include/hal/gpio_types.h

**Macros**

`GPIO_PIN_REG_0`

`GPIO_PIN_REG_1`
| GPIO_PIN_REG 25 |
| GPIO_PIN_REG 26 |
| GPIO_PIN_REG 27 |
| GPIO_PIN_REG 28 |
| GPIO_PIN_REG 29 |
| GPIO_PIN_REG 30 |
| GPIO_PIN_REG 31 |
| GPIO_PIN_REG 32 |
| GPIO_PIN_REG 33 |
| GPIO_PIN_REG 34 |
| GPIO_PIN_REG 35 |
| GPIO_PIN_REG 36 |
| GPIO_PIN_REG 37 |
| GPIO_PIN_REG 38 |
| GPIO_PIN_REG 39 |
| GPIO_PIN_REG 40 |
| GPIO_PIN_REG 41 |
| GPIO_PIN_REG 42 |
| GPIO_PIN_REG 43 |
| GPIO_PIN_REG 44 |
| GPIO_PIN_REG 45 |
| GPIO_PIN_REG 46 |
| GPIO_PIN_REG 47 |
Chapter 2. API 参考

GPIO_PIN_REG_48

Enumerations

defined enum gpio_port_t

    Values:

    enumerator GPIO_PORT_0

    enumerator GPIO_PORT_MAX

defined enum gpio_num_t

    Values:

    enumerator GPIO_NUM_NC
        Use to signal not connected to S/W

    enumerator GPIO_NUM_0
        GPIO0, input and output

    enumerator GPIO_NUM_1
        GPIO1, input and output

    enumerator GPIO_NUM_2
        GPIO2, input and output

    enumerator GPIO_NUM_3
        GPIO3, input and output

    enumerator GPIO_NUM_4
        GPIO4, input and output

    enumerator GPIO_NUM_5
        GPIO5, input and output

    enumerator GPIO_NUM_6
        GPIO6, input and output

    enumerator GPIO_NUM_7
        GPIO7, input and output

    enumerator GPIO_NUM_8
        GPIO8, input and output

    enumerator GPIO_NUM_9
        GPIO9, input and output

    enumerator GPIO_NUM_10
        GPIO10, input and output
enumerator GPIO_NUM_11
    GPIO11, input and output

enumerator GPIO_NUM_12
    GPIO12, input and output

enumerator GPIO_NUM_13
    GPIO13, input and output

enumerator GPIO_NUM_14
    GPIO14, input and output

enumerator GPIO_NUM_15
    GPIO15, input and output

enumerator GPIO_NUM_16
    GPIO16, input and output

enumerator GPIO_NUM_17
    GPIO17, input and output

enumerator GPIO_NUM_18
    GPIO18, input and output

enumerator GPIO_NUM_19
    GPIO19, input and output

enumerator GPIO_NUM_20
    GPIO20, input and output

enumerator GPIO_NUM_21
    GPIO21, input and output

enumerator GPIO_NUM_22
    GPIO22, input and output

enumerator GPIO_NUM_23
    GPIO23, input and output

enumerator GPIO_NUM_25
    GPIO25, input and output

enumerator GPIO_NUM_26
    GPIO26, input and output

enumerator GPIO_NUM_27
    GPIO27, input and output
enumerator GPIO_NUM_28
GPIO28, input and output

enumerator GPIO_NUM_29
GPIO29, input and output

enumerator GPIO_NUM_30
GPIO30, input and output

enumerator GPIO_NUM_31
GPIO31, input and output

enumerator GPIO_NUM_32
GPIO32, input and output

enumerator GPIO_NUM_33
GPIO33, input and output

enumerator GPIO_NUM_34
GPIO34, input mode only

enumerator GPIO_NUM_35
GPIO35, input mode only

enumerator GPIO_NUM_36
GPIO36, input mode only

enumerator GPIO_NUM_37
GPIO37, input mode only

enumerator GPIO_NUM_38
GPIO38, input mode only

enumerator GPIO_NUM_39
GPIO39, input mode only

enumerator GPIO_NUM_MAX

enum gpio_int_type_t

Values:

enumerator GPIO_INTR_DISABLE
GPIO interrupt

enumerator GPIO_INTR_POSEDGE
GPIO interrupt type: rising edge

enumerator GPIO_INTR_NEGEDGE
GPIO interrupt type: falling edge
enumerator GPIO_INTR_ANYEDGE
   GPIO interrupt type: both rising and falling edge

enumerator GPIO_INTR_LOW_LEVEL
   GPIO interrupt type: input low level trigger

enumerator GPIO_INTR_HIGH_LEVEL
   GPIO interrupt type: input high level trigger

enumerator GPIO_INTR_MAX

enum gpio_mode_t
   Values:

enumerator GPIO_MODE_DISABLE
   GPIO mode: disable input and output

enumerator GPIO_MODE_INPUT
   GPIO mode: input only

enumerator GPIO_MODE_OUTPUT
   GPIO mode: output only mode

enumerator GPIO_MODE_OUTPUT_OD
   GPIO mode: output only with open-drain mode

enumerator GPIO_MODE_INPUT_OUTPUT_OD
   GPIO mode: output and input with open-drain mode

enumerator GPIO_MODE_INPUT_OUTPUT
   GPIO mode: output and input mode

enum gpio_pullup_t
   Values:

enumerator GPIO_PULLUP_DISABLE
   Disable GPIO pull-up resistor

enumerator GPIO_PULLUP_ENABLE
   Enable GPIO pull-up resistor

enum gpio pulldown t
   Values:

enumerator GPIO_PULLDOWN_DISABLE
   Disable GPIO pull-down resistor

enumerator GPIO_PULLDOWN_ENABLE
   Enable GPIO pull-down resistor
enum gpio_pull_mode_t

Values:

enumerator GPIO_PULLUP_ONLY
    Pad pull up

enumerator GPIO_PULLDOWN_ONLY
    Pad pull down

enumerator GPIO_PULLUP_PULLDOWN
    Pad pull up + pull down

enumerator GPIO_FLOATING
    Pad floating

denum gpio_drive_cap_t

Values:

enumerator GPIO_DRIVE_CAP_0
    Pad drive capability: weak

enumerator GPIO_DRIVE_CAP_1
    Pad drive capability: stronger

enumerator GPIO_DRIVE_CAP_2
    Pad drive capability: medium

enumerator GPIO_DRIVE_CAP_DEFAULT
    Pad drive capability: medium

enumerator GPIO_DRIVE_CAP_3
    Pad drive capability: strongest

enumerator GPIO_DRIVE_CAP_MAX

API 参考 - RTC GPIO

Header File

- components/driver/include/driver/rtc_io.h

Functions

bool rtc_gpio_is_valid_gpio (gpio_num_t gpio_num)

Determine if the specified GPIO is a valid RTC GPIO.

参数 gpio_num - GPIO number
返回 true if GPIO is valid for RTC GPIO use. false otherwise.

int rtc_io_number_get (gpio_num_t gpio_num)

Get RTC IO index number by gpio number.

参数 gpio_num - GPIO number
**Chapter 2. API 参考**

### esp_err_t rtc_gpio_init (gpio_num_t gpio_num)

Init a GPIO as RTC GPIO.

- **参数** `gpio_num` - GPIO number (e.g. GPIO_NUM_12)
- **返回**
  - ESP_OK success
  - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

### esp_err_t rtc_gpio_deinit (gpio_num_t gpio_num)

Init a GPIO as digital GPIO.

- **参数** `gpio_num` - GPIO number (e.g. GPIO_NUM_12)
- **返回**
  - ESP_OK success
  - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

### uint32_t rtc_gpio_get_level (gpio_num_t gpio_num)

Get the RTC IO input level.

- **参数** `gpio_num` - GPIO number (e.g. GPIO_NUM_12)
- **返回**
  - 1 High level
  - 0 Low level
  - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

### esp_err_t rtc_gpio_set_level (gpio_num_t gpio_num, uint32_t level)

Set the RTC IO output level.

- **参数**
  - `gpio_num` - GPIO number (e.g. GPIO_NUM_12)
  - `level` - output level
- **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

### esp_err_t rtc_gpio_set_direction (gpio_num_t gpio_num, rtc_gpio_mode_t mode)

RTC GPIO set direction.

Configure RTC GPIO direction, such as output only, input only, output and input.

- **参数**
  - `gpio_num` - GPIO number (e.g. GPIO_NUM_12)
  - `mode` - GPIO direction
- **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

### esp_err_t rtc_gpio_set_direction_in_sleep (gpio_num_t gpio_num, rtc_gpio_mode_t mode)

RTC GPIO set direction in deep sleep mode or disable sleep status (default). In some application scenarios, IO needs to have another states during deep sleep.

**NOTE:** ESP32 support INPUT_ONLY mode. ESP32S2 support INPUT_ONLY, OUTPUT ONLY, INPUT_OUTPUT mode.

- **参数**
  - `gpio_num` - GPIO number (e.g. GPIO_NUM_12)
  - `mode` - GPIO direction
- **返回**
  - ESP_OK Success
  - ESP_ERR_INVALID_ARG GPIO is not an RTC IO
esp_err_t rtc_gpio_pullup_en (gpio_num_t gpio_num)
RTC GPIO pullup enable.

This function only works for RTC IOs. In general, call gpio_pullup_en, which will work both for normal
GPIOs and RTC IOs.

参数 gpio_num - GPIO number (e.g. GPIO_NUM_12)

返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t rtc_gpio_pulldown_en (gpio_num_t gpio_num)
RTC GPIO pulldown enable.

This function only works for RTC IOs. In general, call gpio_pulldown_en, which will work both for normal
GPIOs and RTC IOs.

参数 gpio_num - GPIO number (e.g. GPIO_NUM_12)

返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t rtc_gpio_pullup_dis (gpio_num_t gpio_num)
RTC GPIO pullup disable.

This function only works for RTC IOs. In general, call gpio_pullup_dis, which will work both for normal
GPIOs and RTC IOs.

参数 gpio_num - GPIO number (e.g. GPIO_NUM_12)

返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t rtc_gpio_pulldown_dis (gpio_num_t gpio_num)
RTC GPIO pulldown disable.

This function only works for RTC IOs. In general, call gpio_pulldown_dis, which will work both for normal
GPIOs and RTC IOs.

参数 gpio_num - GPIO number (e.g. GPIO_NUM_12)

返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t rtc_gpio_set_drive_capability (gpio_num_t gpio_num, gpio_drive_cap_t strength)
Set RTC GPIO pad drive capability.

参数
• gpio_num - GPIO number, only support output GPIOs
• strength - Drive capability of the pad

返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error

esp_err_t rtc_gpio_get_drive_capability (gpio_num_t gpio_num, gpio_drive_cap_t *strength)
Get RTC GPIO pad drive capability.

参数
• gpio_num - GPIO number, only support output GPIOs
• strength - Pointer to accept drive capability of the pad

返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error
**esp_err_t rtc_gpio_hold_en(gpio_num_t gpio_num)**
Enable hold function on an RTC IO pad.

Enabling HOLD function will cause the pad to latch current values of input enable, output enable, output value, function, drive strength values. This function is useful when going into light or deep sleep mode to prevent the pin configuration from changing.

参数 gpio_num - GPIO number (e.g. GPIO_NUM_12)
返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

**esp_err_t rtc_gpio_hold_dis(gpio_num_t gpio_num)**
Disable hold function on an RTC IO pad.

Disabling hold function will allow the pad receive the values of input enable, output enable, output value, function, drive strength from RTC_IO peripheral.

参数 gpio_num - GPIO number (e.g. GPIO_NUM_12)
返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

**esp_err_t rtc_gpio_isolate(gpio_num_t gpio_num)**
Helper function to disconnect internal circuits from an RTC IO. This function disables input, output, pullup, pulldown, and enables hold feature for an RTC IO. Use this function if an RTC IO needs to be disconnected from internal circuits in deep sleep, to minimize leakage current.

In particular, for ESP32-WROVER module, call rtc_gpio_isolate(GPIO_NUM_12) before entering deep sleep, to reduce deep sleep current.

参数 gpio_num - GPIO number (e.g. GPIO_NUM_12).
返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if GPIO is not an RTC IO

**esp_err_t rtc_gpio_force_hold_en_all(void)**
Enable force hold signal for all RTC IOs.

Each RTC pad has a “force hold” input signal from the RTC controller. If this signal is set, pad latches current values of input enable, function, output enable, and other signals which come from the RTC mux. Force hold signal is enabled before going into deep sleep for pins which are used for EXT1 wakeup.

**esp_err_t rtc_gpio_force_hold_dis_all(void)**
Disable force hold signal for all RTC IOs.

**esp_err_t rtc_gpio_wakeup_enable(gpio_num_t gpio_num, gpio_int_type_t intr_type)**
Enable wakeup from sleep mode using specific GPIO.

参数
- gpio_num - GPIO number
- intr_type - Wakeup on high level (GPIO_INTR_HIGH_LEVEL) or low level (GPIO_INTR_LOW_LEVEL)

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO, or intr_type is not one of GPIO_INTR_HIGH_LEVEL, GPIO_INTR_LOW_LEVEL.

**esp_err_t rtc_gpio_wakeup_disable(gpio_num_t gpio_num)**
Disable wakeup from sleep mode using specific GPIO.

参数 gpio_num - GPIO number
返回
- ESP_OK on success
Chapter 2. API Reference

- ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO

**Macros**

```c
RTC_GPIO_IS_VALID_GPIO(gpio_num)
```

**Header File**

- components/hal/include/hal/rtc_io_types.h

**Enumerations**

```c
enum rtc_gpio_mode_t
{
    RTC_GPIO_MODE_INPUT_ONLY,
    RTC_GPIO_MODE_OUTPUT_ONLY,
    RTC_GPIO_MODE_INPUT_OUTPUT,
    RTC_GPIO_MODE_DISABLED,
    RTC_GPIO_MODE_OUTPUT_OD,
    RTC_GPIO_MODE_INPUT_OUTPUT_OD,
};
```

2.6.7 通用定时器

**简介**

通用定时器是 ESP32 定时器组外设的驱动程序。ESP32 硬件定时器分辨率高，具有灵活的报警功能。定时器内部计数器达到特定目标数值的行为被称为定时器报警。定时器报警时将调用用户注册的不同定时器回调函数。

通用定时器通常在以下场景中使用：
- 同时挂钟一般自由运行，随时随地获取高分辨率时间戳；
- 生成周期性警报，定期触发事件；
- 生成一次性警报，在目标时间内响应。
功能概述

下文介绍了配置和操作定时器的常规步骤:

- 资源分配 - 获取定时器句柄应设置的参数，以及如何在通用定时器完成工作时回收资源。
- 设置和获取计数值 - 如何强制定时器从起点开始计数，以及如何随时获取计数值。
- 设置报警动作 - 启动警报事件应设置的参数。
- 注册事件回调函数 - 如何将用户的特定代码挂接到警报事件回调函数。
- 使能和禁用定时器 - 如何使能和禁用定时器。
- 启动和停止定时器 - 通过不同报警行为启动定时器的典型使用场景。
- 电源管理 - 选择不同的时钟源将会如何影响功耗。
- IRAM 安全 - 在 cache 禁用的情况下，如何更好地让定时器处理中断事务以及实现 IO 控制功能。
- 线程安全 - 驱动程序保证哪些 API 线程安全。
- Kconfig 选项 - 支持的 Kconfig 选项，这些选项会对驱动程序行为产生不同影响。

资源分配

不同的 ESP 芯片可能有不同数量的独立定时器组，每组内也可能有若干个独立定时器。1

通用定时器实例由 `gptimer_handle_t` 表示。后台驱动会在资源池中管理所有可用的硬件资源，这样便于无需考虑硬件所属的定时器以及定时器组。

要安装一个定时器实例，需要提前提供配置结构体 `gptimer_config_t`:

```c
.gptimer_config_t timer_config = {
  .clk_src = GPTIMER_CLK_SRC_DEFAULT,
  .direction = GPTIMER_COUNT_UP,
  .resolution_hz = 1 * 1000 * 1000, // 1MHz, 1 tick = 1us
};
```

完成上述结构配置之后，可以将结构传递给 `gptimer_new_timer()`，用以实例化定时器实例并返回定时器句柄。

该函数可能由于内存不足、参数无效或错误而失败。具体来说，当没有更多的空闲定时器（即所有硬件资源已用完）时，将返回 ESP_ERR_NOT_FOUND。可用定时器总数由 SOC_TIMER_GROUP_TOTAL_TIMERS 表示，不同的 ESP 芯片该数值不同。

如已不再需要之前创建的通用定时器实例，应通过调用 `gptimer_del_timer()` 回收定时器，以便底层硬件定时器用于其他目的。在删除通用定时器句柄之前，请通过 `gptimer_disable()` 禁用定时器，或者通过 `gptimer_enable()` 确定定时器尚未使能。

创建分辨率为 1 MHz 的通用定时器句柄

```c
gptimer_handle_t gtimer = NULL;
gptimer_config_t timer_config = {
  .clk_src = GPTIMER_CLK_SRC_DEFAULT,
  .direction = GPTIMER_COUNT_UP,
  .resolution_hz = 1 * 1000 * 1000, // 1MHz, 1 tick = 1us
};
ESP_ERROR_CHECK(gptimer_new_timer(&timer_config, &gtimer));
```

设置和获取计数值

创建通用定时器时，内部计数器将默认重置为零。计数值可以通过 `gptimer_set_raw_count()` 异步更新。最大计数值取决于硬件定时器的位宽，也会在 SOC 宏 SOC_TIMER_GROUP_COUNTER_BIT_WIDTH 中有所反映。当更新活动定时器的原始计数值时，定时器将立即重新启动计数。

1 不同 ESP 芯片系列的通用定时器实例数量可能不同。了解详细信息，请参考《ESP32 技术参考手册》> 章节定时器组 (TIMG) (PDF)。驱动程序不会禁止您申请更多的定时器，但是当所有可用的硬件资源用完时将会返回错误。在分配资源时，请务必检查返回值（例如 `gptimer_new_timer()`）。
### Chapter 2. API 参考

计数值可以随时通过 gptimer_get_raw_count() 获取。

#### 设置警报动作

对于大多数通用定时器使用场景而言，应在启动定时器之前设置警报动作，但不包括简单的挂钟场景，该场景仅需自由运行的定时器。设置警报动作，需要根据如何使用警报事件来配置 gptimer_alarm_config_t 的不同参数:

- gptimer_alarm_config_t::alarm_count 设置触发警报事件的目标计数值。设置警报值时还需考虑计数方向。尤其是当 gptimer_alarm_config_t::auto_reload_on_alarm 为 true 时，gptimer_alarm_config_t::alarm_count 和 gptimer Alarm_config_t::reload_count 不能设置为相同的值，因为警报值和重载值相同时没有意义。
- gptimer_alarm_config_t::reload_count 代表警报事件发生时要重载的计数值。此配置仅在 gptimer_alarm_config_t::auto_reload_on_alarm 设置为 true 时生效。
- gptimer Alarm_config_t::auto_reload_on_alarm 标志设置是否使能自动重载功能。如果使能，硬件定时器将在警报事件发生时立即将 gptimer alarm config_t::reload_count 的值重载到计数器中。

要使警报配置生效，需要调用 gptimer_set_alarm_action()。特别地，当 gptimer_alarm_config_t 设置为 NULL 时，报警功能将被禁用。

#### 备注：

如果警报值已设置且定时器超过该值，则会立即触发警报。

#### 注册事件回调函数

定时器启动后，可动态产生特定事件（如“警报事件”）。如需在事件发生时调用某些函数，可以通过 gptimer_register_event_callbacks() 将函数挂载到中断服务例程 (ISR)。gptimer_event_callbacks_t 中列出了所有支持的事件回调函数:

- gptimer_event_callbacks_t::on_alarm 设置警报事件的回调函数。由于此函数在 ISR 上下文中调用，必须确保该函数不会试图阻塞（例如，确保仅从函数内调用具有 ISR 后缀的 FreeRTOS API）。函数原型在 gptimer_alarm_config_t::reload_count 中有所声明。

您也可以通过参数 user_data 将自己的上下文保存到 gptimer_register_event_callbacks() 中，用户数据将直接传递给回调函数。

此功能将为定时器延迟安装中断服务，但不使能中断服务。所以，请在 gptimer_enable() 之前调用这一函数，否则将返回 ESP_ERR_INVALID_STATE 错误。了解详细信息，请查看章节 使能和禁用定时器。

#### 使能和禁用定时器

在对定时器进行 IO 控制之前，需要先调用 gptimer_enable() 使能定时器。此函数功能如下:

- 此函数将把定时器驱动程序的状态从 init 切换为 enable。
- 如果 gptimer_register_event_callbacks() 已经延迟安装中断服务，此函数将使能中断服务。
- 如果选择了特定的时钟源（例如 APB 时钟），此函数将获取适当的电源管理器，了解更多信息，请查看章节 电源管理。

调用 gptimer_disable() 会进行相反的操作，即将定时器驱动程序恢复到 init 状态，禁用中断服务并释放电源管理器。

#### 启动和停止定时器

启动和停止是定时器的基本 IO 操作。调用 gptimer_start() 可以使内部计数器开始工作，而 gptimer_stop() 可以使计数器停止工作。下文说明了如何在存在或不存在警报事件的情况下启动定时器。

#### 将定时器作为挂钟启动

```c
ESP_OK = gptimer_enable(gptimer);  // 使能定时器
ESP_OK = gptimer_start(gptimer);   // 开始计数器工作
```

(下页继续)
触发周期性事件

typedef struct {
    uint64_t event_count;
} example_queue_element_t;

static bool example_timer_on_alarm_cb(gptimer_handle_t timer, const gptimer_alarm_event_data_t *edata, void *user_ctx) {
    BaseType_t high_task_awoken = pdFALSE;
    QueueHandle_t queue = (QueueHandle_t)user_ctx;
    // Retrieve the count value from event data
    example_queue_element_t ele = {
        .event_count = edata->count_value
    };
    // Optional: send the event data to other task by OS queue
    // Don't introduce complex logics in callbacks
    // Suggest dealing with event data in the main loop, instead of in this callback
    xQueueSendFromISR(queue, &ele, &high_task_awoken);
    // return whether we need to yield at the end of ISR
    return high_task_awoken == pdTRUE;
}

uint64_t count;
ESP_ERROR_CHECK(gptimer_get_raw_count(gptimer, &count));

typedef struct {
    uint64_t event_count;
} example_queue_element_t;

static bool example_timer_on_alarm_cb(gptimer_handle_t timer, const gptimer_alarm_event_data_t *edata, void *user_ctx) {
    BaseType_t high_task_awoken = pdFALSE;
    QueueHandle_t queue = (QueueHandle_t)user_ctx;
    // Stop timer the sooner the better
    gptimer_stop(timer);
    // Retrieve the count value from event data
    example_queue_element_t ele = {
        .event_count = edata->count_value
    };
    // Optional: send the event data to other task by OS queue
    xQueueSendFromISR(queue, &ele, &high_task_awoken);
}

gptimer_alarm_config_t alarm_config = {
    .reload_count = 0, // counter will reload with 0 on alarm event
    .alarm_count = 1000000, // period = 1s @resolution 1MHz
    .flags.auto_reload_on_alarm = true, // enable auto-reload
};
ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));

gptimer_event_callbacks_t cbs = {
    .on_alarm = example_timer_on_alarm_cb, // register user callback
};
ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
ESP_ERROR_CHECK(gptimer_enable(gptimer));
ESP_ERROR_CHECK(gptimer_start(gptimer));

触发一次性事件

typedef struct {
    uint64_t event_count;
} example_queue_element_t;

static bool example_timer_on_alarm_cb(gptimer_handle_t timer, const gptimer_alarm_event_data_t *edata, void *user_ctx) {
    BaseType_t high_task_awoken = pdFALSE;
    QueueHandle_t queue = (QueueHandle_t)user_ctx;
    // Stop timer the sooner the better
    gptimer_stop(timer);
    // Retrieve the count value from event data
    example_queue_element_t ele = {
        .event_count = edata->count_value
    };
    // Optional: send the event data to other task by OS queue
    xQueueSendFromISR(queue, &ele, &high_task_awoken);
}
```c
// return whether we need to yield at the end of ISR
return high_task_awoken == pdTRUE;
}
gptimer_alarm_config_t alarm_config = {
    .alarm_count = 1 * 1000 * 1000, // alarm target = 1s @resolution 1MHz
};
ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));
gptimer_event_callbacks_t cbs = {
    .on_alarm = example_timer_on_alarm_cb, // register user callback
};
ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
ESP_ERROR_CHECK(gptimer_enable(gptimer));
ESP_ERROR_CHECK(gptimer_start(gptimer));
```

### 警报值动态更新
通过更改 `gptimer_alarm_event_data_t::alarm_value`，可以在 ISR 程序回调中动态更新警报值。警报值将在回调函数返回后更新。

```c
typedef struct {
    uint64_t event_count;
} example_queue_element_t;

static bool example_timer_on_alarm_cb(gptimer_handle_t timer, const gptimer_alarm_event_data_t *edata, void *user_ctx)
{
    BaseType_t high_task_awoken = pdFALSE;
    QueueHandle_t queue = (QueueHandle_t)user_data;
    // Retrieve the count value from event data
    example_queue_element_t ele = {
        .event_count = edata->count_value
    };
    // Optional: send the event data to other task by OS queue
    xQueueSendFromISR(queue, &ele, &high_task_awoken);
    // reconfigure alarm value
    gptimer_alarm_config_t alarm_config = {
        .alarm_count = edata->alarm_value + 1000000, // alarm in next 1s
    };
    gptimer_set_alarm_action(timer, &alarm_config);
    // return whether we need to yield at the end of ISR
    return high_task_awoken == pdTRUE;
}
gptimer_alarm_config_t alarm_config = {
    .alarm_count = 1000000, // initial alarm target = 1s @resolution 1MHz
};
ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));
gptimer_event_callbacks_t cbs = {
    .on_alarm = example_timer_on_alarm_cb, // register user callback
};
ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
ESP_ERROR_CHECK(gptimer_enable(gptimer));
ESP_ERROR_CHECK(gptimer_start(gptimer));
```

### 电源管理
当使能电源管理时（即 `CONFIG_PM_ENABLE` 已打开），系统将在进入 Light-sleep 模式之前调整 APB 频率，从而可能会改变通用定时器的计数步骤周期，导致计时不准确。

然而，驱动程序可以通过获取类型为 `ESP_PM_APB_FREQ_MAX` 的电源管理锁来阻止系统更改 APB 频率。

---

**Espressif Systems 1024**

*Release v5.1-dev-2066-g7869f4e151*

[Submit Document Feedback]
每当驱动程序创建一个通用定时器实例，且该实例选择`GPTIMER_CLK_SRC_APB`作为其时钟源时，驱动程序会确保在通过`gptimer_enable()`使能定时器时，已经获取了电源管理锁。同样，当为该定时器调用`gptimer_disable()`时，驱动程序会释放电源管理锁。

如果选择`GPTIMER_CLK_SRC_XTAL`等其他时钟源，那么驱动程序不会安装电源管理锁。只要时钟源仍可提供足够的分辨率，XTAL 时钟源就更适合低功耗应用。

**IRAM 安全** 默认情况下，当 cache 因写入或擦除 flash 等原因而被禁用时，通用定时器的中断服务将延迟，造成警报中断无法及时执行。在实时应用程序中通常需要避免这一情况发生。

调用 Kconfig 选项`CONFIG_GPTIMER_ISR_IRAM_SAFE`可实现如下功能：
- 即使禁用 cache 也可使能正在运行的中断
- 将 ISR 使用的所有函数放入 IRAM
- 将驱动程序对象放入 DRAM（以防意外映射到 PSRAM）

这将允许中断在 cache 禁用时运行，但会增加 IRAM 使用量。

调用另一 Kconfig 选项`CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM`也可将常用的 IO 控制功能放入 IRAM，以便这些函数在 cache 禁用时也能执行。常用的 IO 控制功能如下：
- `gptimer_start()`
- `gptimer_stop()`
- `gptimer_get_raw_count()`
- `gptimer_set_raw_count()`
- `gptimer_set_alarm_action()`

**线程安全** 驱动程序会保证工厂函数`gptimer_new_timer()`的线程安全，这意味着您可以从不同的RTOS 任务中调用这一函数，而无需额外的锁保护。

由于驱动程序通过使用临界区来防止这些函数在任务和 ISR 中同时被调用，所以以下函数能够在 ISR 上下文中运行。
- `gptimer_start()`
- `gptimer_stop()`
- `gptimer_get_raw_count()`
- `gptimer_set_raw_count()`
- `gptimer_set_alarm_action()`

将`gptimer_handle_t`作为第一个位置参数的其他函数不被视作线程安全，也就是说应该避免从多个任务中调用这些函数。

**Kconfig 选项**
- `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` 控制放置通用定时器控制函数（IRAM 或 flash）的位置。了解更多信息，请参考章节`IRAM 安全`。
- `CONFIG_GPTIMER_ISR_IRAM_SAFE` 控制默认 ISR 程序在 cache 禁用时是否可以运行。了解更多信息，请参考章节`IRAM 安全`。
- `CONFIG_GPTIMER_ENABLE_DEBUG_LOG` 用于启用调试日志输出。启用这一选项将增加固件二进制文件大小。

**应用示例**
- 示例`peripherals/timer_group/gptimer`中列出了通用定时器的典型用例。

\[^2\]`gptimer_event_callbacks_t::on_alarm`回调函数和这一函数调用的函数也需放在 IRAM 中，请自行处理。
Chapter 2. API 参考

API 参考

Header File

- components/driver/include/driver/gptimer.h

Functions

gptimer_new_timer (const gptimer_config_t *config, gptimer_handle_t *ret_timer)

Create a new General Purpose Timer, and return the handle.

备注: The newly created timer is put in the init state.

参数
- config [in] GPTimer configuration
- ret_timer [out] Returned timer handle

返回
- ESP_OK: Create GPTimer successfully
- ESP_ERR_INVALID_ARG: Create GPTimer failed because of invalid argument
- ESP_ERR_NO_MEM: Create GPTimer failed because out of memory
- ESP_ERR_NOT_FOUND: Create GPTimer failed because all hardware timers are used up and no more free one
- ESP_FAIL: Create GPTimer failed because of other error

gptimer_del_timer (gptimer_handle_t timer)

Delete the GPTimer handle.

备注: A timer can’t be in the enable state when this function is invoked. See also gptimer_disable for how to disable a timer.

参数 timer [in] Timer handle created by gptimer_new_timer

返回
- ESP_OK: Delete GPTimer successfully
- ESP_ERR_INVALID_ARG: Delete GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete GPTimer failed because the timer is not in init state
- ESP_FAIL: Delete GPTimer failed because of other error

gptimer_set_raw_count (gptimer_handle_t timer, uint64_t value)

Set GPTimer raw count value.

备注: When updating the raw count of an active timer, the timer will immediately start counting from the new value.

备注: This function is allowed to run within ISR context

备注: This function is allowed to be executed when Cache is disabled, by enabling CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM

参数
- timer [in] Timer handle created by gptimer_new_timer
Chapter 2. API

• **value** - [in] Count value to be set

返回
• ESP_OK: Set GPTimer raw count value successfully
• ESP_ERR_INVALID_ARG: Set GPTimer raw count value failed because of invalid argument
• ESP_FAIL: Set GPTimer raw count value failed because of other error

```c
esp_err_t gptimer_get_raw_count (gptimer_handle_t timer, uint64_t *value)
```

Get GPTimer raw count value.

备注: This function will trigger a software capture event and then return the captured count value.

备注: With the raw count value and the resolution set in the gptimer_config_t, you can convert the count value into seconds.

备注: This function is allowed to run within ISR context

备注: This function is allowed to be executed when Cache is disabled, by enabling CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM

参数
• **timer** - [in] Timer handle created by gptimer_new_timer
• **value** - [out] Returned GPTimer count value

返回
• ESP_OK: Get GPTimer raw count value successfully
• ESP_ERR_INVALID_ARG: Get GPTimer raw count value failed because of invalid argument
• ESP_FAIL: Get GPTimer raw count value failed because of other error

```c
esp_err_t gptimer_get_captured_count (gptimer_handle_t timer, uint64_t *value)
```

Get GPTimer captured count value.

备注: The capture action can be issued either by external event or by software (see also gptimer_get_raw_count).

备注: This function is allowed to run within ISR context

备注: This function is allowed to be executed when Cache is disabled, by enabling CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM

参数
• **timer** - [in] Timer handle created by gptimer_new_timer
• **value** - [out] Returned captured count value

返回
• ESP_OK: Get GPTimer captured count value successfully
• ESP_ERR_INVALID_ARG: Get GPTimer captured count value failed because of invalid argument
**Chapter 2. API 参考**

- **ESP_FAIL**: Get GPTimer captured count value failed because of other error

```c
esp_err_t gptimer_register_event_callbacks(gptimer_handle_t timer, const gptimer_event_callbacks_t *cbs, void *user_data)
```

Set callbacks for GPTimer.

**备注**: User registered callbacks are expected to be runnable within ISR context

**备注**: The first call to this function needs to be before the call to `gptimer_enable`

**备注**: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

### 参数
- `timer` **[in]** Timer handle created by `gptimer_new_timer`
- `cbs` **[in]** Group of callback functions
- `user_data` **[in]** User data, which will be passed to callback functions directly

### 返回
- **ESP_OK**: Set event callbacks successfully
- **ESP_ERR_INVALID_ARG**: Set event callbacks failed because of invalid argument
- **ESP_ERR_INVALID_STATE**: Set event callbacks failed because the timer is not in init state
- **ESP_FAIL**: Set event callbacks failed because of other error

```c
esp_err_t gptimer_set_alarm_action(gptimer_handle_t timer, const gptimer_alarm_config_t *config)
```

Set alarm event actions for GPTimer.

**备注**: This function is allowed to run within ISR context, so that user can set new alarm action immediately in the ISR callback.

**备注**: This function is allowed to be executed when Cache is disabled, by enabling `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM`

### 参数
- `timer` **[in]** Timer handle created by `gptimer_new_timer`
- `config` **[in]** Alarm configuration, especially, set config to NULL means disabling the alarm function

### 返回
- **ESP_OK**: Set alarm action for GPTimer successfully
- **ESP_ERR_INVALID_ARG**: Set alarm action for GPTimer failed because of invalid argument
- **ESP_FAIL**: Set alarm action for GPTimer failed because of other error

```c
esp_err_t gptimer_enable(gptimer_handle_t timer)
```

Enable GPTimer.

**备注**: This function will transit the timer state from init to enable.
**备 注:** This function will enable the interrupt service, if it’s lazy installed in `gptimer_register_event_callbacks`.

**备 注:** This function will acquire a PM lock, if a specific source clock (e.g. APB) is selected in the `gptimer_config_t`, while `CONFIG_PM_ENABLE` is enabled.

**备 注:** Enable a timer doesn’t mean to start it. See also `gptimer_start` for how to make the timer start counting.

**参数** `timer` -[in] Timer handle created by `gptimer_new_timer`

**返回**
- ESP_OK: Enable GPTimer successfully
- ESP_ERR_INVALID_ARG: Enable GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable GPTimer failed because the timer is already enabled
- ESP_FAIL: Enable GPTimer failed because of other error

`esp_err_t` `gptimer_disable` (*gptimer_handle_t* `timer`)

Disable GPTimer.

**备 注:** This function will do the opposite work to the `gptimer_enable`.

**备 注:** Disable a timer doesn’t mean to stop it. See also `gptimer_stop` for how to make the timer stop counting.

**参数** `timer` -[in] Timer handle created by `gptimer_new_timer`

**返回**
- ESP_OK: Disable GPTimer successfully
- ESP_ERR_INVALID_ARG: Disable GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Disable GPTimer failed because the timer is not enabled yet
- ESP_FAIL: Disable GPTimer failed because of other error

`esp_err_t` `gptimer_start` (*gptimer_handle_t* `timer`)

Start GPTimer (internal counter starts counting)

**备 注:** This function should be called when the timer is in the enable state (i.e. after calling `gptimer_enable`)

**备 注:** This function is allowed to run within ISR context.

**备 注:** This function will be placed into IRAM if `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is on, so that it’s allowed to be executed when Cache is disabled.

**参数** `timer` -[in] Timer handle created by `gptimer_new_timer`


Chapter 2. API 参考

返回
- ESP_OK: Start GPTimer successfully
- ESP_ERR_INVALID_ARG: Start GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Start GPTimer failed because the timer is not enabled yet
- ESP_FAIL: Start GPTimer failed because of other error

\[ \text{esp_err_t gptimer_stop (gptimer_handle_t timer)} \]

Stop GPTimer (internal counter stops counting)

备注: This function should be called when the timer is in the enable state (i.e. after calling gptimer_enable)

备注: This function is allowed to run within ISR context

备注: This function will be placed into IRAM if CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM is on, so that it’s allowed to be executed when Cache is disabled

参数 timer - [in] Timer handle created by gptimer_new_timer

返回
- ESP_OK: Stop GPTimer successfully
- ESP_ERR_INVALID_ARG: Stop GPTimer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Stop GPTimer failed because the timer is not enabled yet
- ESP_FAIL: Stop GPTimer failed because of other error

Structures

\[ \text{struct gptimer_config_t} \]

General Purpose Timer configuration.

Public Members

\[ gptimer_clock_source_t \text{clk_src} \]

GPTimer clock source

\[ gptimer_count_direction_t \text{direction} \]

Count direction

uint32_t \text{resolution_hz} \]

Counter resolution (working frequency) in Hz, hence, the step size of each count tick equals to \((1 / \text{resolution_hz})\) seconds

uint32_t \text{intr_shared} \]

Set true, the timer interrupt number can be shared with other peripherals

struct gptimer_config_t::[anonymous] \text{flags} \]

GPTimer config flags
Chapter 2. API 参考

```c
struct gptimer_event_callbacks_t
    Group of supported GPTimer callbacks.
```

备注: The callbacks are all running under ISR environment

备注: When CONFIG_GPTIMER_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM.

### Public Members

```c
gptimer_alarm_cb_t on_alarm
    Timer alarm callback
```

```c
struct gptimer_alarm_config_t
    General Purpose Timer alarm configuration.
```

### Public Members

```c
uint64_t alarm_count
    Alarm target count value
```

```c
uint64_t reload_count
    Alarm reload count value, effect only when auto_reload_on_alarm is set to true
```

```c
uint32_t auto_reload_on_alarm
    Reload the count value by hardware, immediately at the alarm event
```

```c
struct gptimer_alarm_config_t::[anonymous] flags
    Alarm config flags
```

### Header File

- components/driver/include/driver/gptimer_etm.h

### Functions

```c
esp_err_t gptimer_new_etm_event (gptimer_handle_t timer, gptimer_etm_event_type_t event_type,
                                esp_etm_event_handle_t*out_event)
```

Get the ETM event for GPTimer.

备注: The created ETM event object can be deleted later by calling esp_etm_del_event.

参数

- timer -[in] Timer handle created by gptimer_new_timer
- event_type -[in] GPTimer ETM event type
- out_event -[out] Returned ETM event handle

返回
Chapter 2. API Reference

- ESP_OK: Get ETM event successfully
- ESP_ERR_INVALID_ARG: Get ETM event failed because of invalid argument
- ESP_FAIL: Get ETM event failed because of other error

```c
esp_err_t gptimer_new_etm_task(gptimer_handle_t timer, gptimer_etm_task_type_t task_type, esp_etm_task_handle_t *out_task)
```

Get the ETM task for GPTimer.

**备注:** The created ETM task object can be deleted later by calling `esp_etm_del_task`

### Parameters
- **timer** [in] Timer handle created by `gptimer_new_timer`
- **task_type** [in] GPTimer ETM task type
- **out_task** [out] Returned ETM task handle

### Return
- ESP_OK: Get ETM task successfully
- ESP_ERR_INVALID_ARG: Get ETM task failed because of invalid argument
- ESP_FAIL: Get ETM task failed because of other error

**Header File**
- `components/driver/include/driver/gptimer_types.h`

**Structures**

```c
struct gptimer_alarm_event_data_t
```

GPTimer alarm event data.

**Public Members**

- **count_value**
  - Current count value
- **alarm_value**
  - Current alarm value

**Type Definitions**

```c
typedef struct gptimer_t *gptimer_handle_t
```

Type of General Purpose Timer handle.

```c
typedef bool (*gptimer_alarm_cb_t)(gptimer_handle_t timer, const gptimer_alarm_event_data_t *edata, void *user_ctx)
```

Timer alarm callback prototype.

- **timer** [in] Timer handle created by `gptimer_new_timer`
- **edata** [in] Alarm event data, fed by driver
- **user_ctx** [in] User data, passed from `gptimer_register_event_callbacks`

**Return** Whether a high priority task has been woken up by this function

**Header File**
- `components/hal/include/hal/timer_types.h`
Chapter 2. API 参考

Type Definitions
typedef soc_periph_gptimer_clk_src_t gptimer_clock_source_t
GPTimer clock source.

备注：User should select the clock source based on the power and resolution requirement

Enumerations
enum gptimer_count_direction_t
GPTimer count direction.
Values:
enumerator GPTIMER_COUNT_DOWN
Decrease count value
enumerator GPTIMER_COUNT_UP
Increase count value

enum gptimer_etm_task_type_t
GPTimer specific tasks that supported by the ETM module.
Values:
enumerator GPTIMER_ETM_TASK_START_COUNT
Start the counter
enumerator GPTIMER_ETM_TASK_STOP_COUNT
Stop the counter
enumerator GPTIMER_ETM_TASK_EN_ALARM
Enable the alarm
enumerator GPTIMER_ETM_TASK_RELOAD
Reload preset value into counter
enumerator GPTIMER_ETM_TASK_CAPTURE
Capture current count value into specific register
enumerator GPTIMER_ETM_TASK_MAX
Maximum number of tasks

enum gptimer_etm_event_type_t
GPTimer specific events that supported by the ETM module.
Values:
enumerator GPTIMER_ETM_EVENT_ALARM_MATCH
Count value matches the alarm target value
enumerator GPTIMER_ETM_EVENT_MAX
Maximum number of events
2.6.8 I2C 驱动程序

概述
I2C 是一种串行同步半双工通信协议，总线上可以同时挂载多个主机和从机。I2C 总线由串行数据线 (SDA) 和串行时钟线 (SCL) 线构成。这些线都需要上拉电阻。

I2C 具有简单且制造成本低廉等优点，主要用于低速外围设备的短距离通信（一英尺以内）。

ESP32 有 2 个 I2C 控制器（也称为端口），负责处理在 I2C 总线上的通信。每个控制器都可以设置为主机或从机。

驱动程序的功能
I2C 驱动程序管理在 I2C 总线上设备的通信，该驱动程序具备以下功能：

- 在主机模式下读写字节
- 支持从机模式
- 读取并写入寄存器，然后由主机读取/写入

使用驱动程序

以下部分将指导您完成 I2C 驱动程序配置和工作的基本步骤：

1. 配置驱动程序 - 设置初始化参数（如主机模式或从机模式、SDA 和 SCL 使用的 GPIO 管脚，时钟速度等）
2. 安装驱动程序 - 激活一个 I2C 控制器的驱动，该控制器可为主机也可为从机
3. 根据是主机还是从机配置驱动程序，选择合适的项目
   a) 主机模式下通信 - 发起通信（主机模式）
   b) 从机模式下通信 - 响应主机消息（从机模式）
4. 中断处理 - 配置 I2C 中断服务
5. 用户自定义配置 - 调整默认的 I2C 通信参数（如时序、位序等）
6. 错误处理 - 如何识别和处理驱动程序配置和通信错误
7. 删除驱动程序 - 在通信结束时释放 I2C 驱动程序所使用的资源

配置驱动程序

建立 I2C 通信第一步是配置驱动程序，这需要设置 i2c_config_t 结构中的几个参数：

- 设置 I2C 工作模式 - 从 i2c_mode_t 中选择主机模式或从机模式
- 设置 通信管脚
  - 指定 SDA 和 SCL 信号使用的 GPIO 管脚
  - 是否启用 ESP32 的内部上拉电阻
- （仅限主机模式）设置 I2C 时钟速度
- （仅限从机模式）设置以下内容:
  - 是否应启用 10 位寻址模式
  - 定义 从机地址

然后，初始化给定 I2C 端口的配置，请使用端口号和 i2c_config_t 作为函数调用参数来调用 i2c_param_config() 函数。

配置示例（主机）:

```c
int i2c_master_port = 0;
i2c_config_t conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = I2C_MASTER_SDA_IO,    // select GPIO specific to your
    .scl_io_num = I2C_MASTER_SCL_IO,
    .expansion_bits = 0,
    .flags = 0,
    .frequency_hz = 100000,
    .operation = I2C_OPERATION_NORMAL,
    .polarity = I2C_POLARITY_LOW,
    .clock_polarity = I2C Clock POLARITY_LOW
};
```
配置示例（从机）：

```c
int i2c_slave_port = I2C_SLAVE_NUM;
i2c_config_t conf_slave = {
    .sda_io_num = I2C_SLAVE_SDA_IO,  // select GPIO specific to your project
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = I2C_SLAVE_SCL_IO,  // select GPIO specific to your project
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .mode = I2C_MODE_SLAVE,
    .slave.addr_10bit_en = 0,
    .slave.slave_addr = ESP_SLAVE_ADDR,  // address of your project
    .clk_flags = 0,
};
```

在此阶段，`i2c_param_config()` 还将其他 I2C 配置参数设置为 I2C 总线协议规范中定义的默认值。有关默认值及修改默认值的详细信息，请参考用户自定义配置。

### 源时钟配置

增加了时钟源分配器，用于支持不同的时钟源。时钟分配器将选择一个满足所有频率和能力要求的时钟源（如 `i2c_config_t::clk_flags` 中的要求）。

当 `i2c_config_t::clk_flags` 为 0 时，时钟分配器将仅根据所需频率进行选择。如果不需要诸如 APB 之类的特殊功能，则可以将时钟分配器配置为仅根据所需频率选择源时钟。为此，请将 `i2c_config_t::clk_flags` 设置为 0。有关时钟特性，请参见下表。

**备注：** 如果时钟不满足请求的功能，则该时钟不是有效的选项，即，请求的功能中的任何位（`clk_flags`）在时钟的功能中均为 0。

### 表 3: ESP32 时钟源特性

<table>
<thead>
<tr>
<th>时钟名称</th>
<th>时钟频率</th>
<th>SCL 的最大频率</th>
<th>时钟功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>APB 时钟</td>
<td>80 MHz</td>
<td>4 MHz</td>
<td>/</td>
</tr>
</tbody>
</table>

对 `i2c_config_t::clk_flags` 的解释如下：

1. `I2C_SCLK_SRC_FLAG_AWARE_DFS`：当 APB 时钟改变时，时钟的波特率不会改变。
2. `I2C_SCLK_SRC_FLAG_LIGHT_SLEEP`：支持轻度睡眠模式，APB 时钟则不支持。
3. ESP32 可能不支持某些标志，请在使用前阅读技术参考手册。

**备注：** 在主机模式下，SCL 的时钟频率不应大于上表中提到的 SCL 的最大频率。
### 备注：SCL 的时钟频率会被上拉电阻和线上电容（或是从机电容）一起影响。因此，用户需要自己选择合适的上拉电阻去保证 SCL 时钟频率是准确的。尽管 I2C 协议推荐上拉电阻值为 1K 欧姆到 10K 欧姆，但是需要根据不同的频率需要选择不同的上拉电阻。

通常来说，所选择的频率越高，需要的上拉电阻越小（但是不要小于 1K 欧姆）。这是因为高电阻会减小电流，这会延长上升时间从而是频率变慢。通常我们推荐的上拉阻值范围为 2K 欧姆到 5K 欧姆，但是用户可能也需要根据他们的实际情况做出一些调整。

### 安装驱动程序

配置好 I2C 驱动程序后，使用以下参数调用函数 `i2c_driver_install()` 安装驱动程序:

- 端口号，从 `i2c_port_t` 中二选一
- 主机或从机模式，从 `i2c_mode_t` 中选择
  - （仅限从机模式）分配用于在从机模式下发送和接收数据的缓存区大小。I2C 是一个以主机为中心的总线，数据只能根据主机的需求从从机传输到主机。因此，从机通常有一个发送缓存区，供从应用程序写入数据使用。数据保存在发送缓存区中，由主机自行读取。
  - 用于分配中断的标志（请参考 `esp_hw_support/include/esp_intr_alloc.h` 中 ESP_INTR_FLAG_* 值）

### 主机模式下通信

安装 I2C 驱动程序后，ESP32 即可与其他 I2C 设备通信。

ESP32 的 I2C 控制器在主机模式下负责与 I2C 从机设备建立通信，并发送命令让从机响应，如进行测量并将结果发给主机。

为优化通信流程，驱动程序提供一个名为“命令链接”的容器，该容器应填充一系列命令，然后传递给 I2C 控制器执行。

### 主机写入数据

下面的示例展示如何为 I2C 主机构建命令链接，从而向从机发送 n 个字节。

![图 7: I2C command link - master write example](image)

下面介绍如何为“主机写入数据”设置命令链接及其内部内容:

1. 使用 `i2c_cmd_link_create()` 创建一个命令链接。
   - 然后，将一系列待发送给从机的数据填充命令链接:
     a) 启动位 - `i2c_master_start()`
     b) 从机地址 - `i2c_master_write_byte()`。提供单字节地址作为调用此函数的实参。
     c) 数据 - 一个或多个字节的数据作为 `i2c_master_write()` 的实参。

---

Espressif Systems 1036 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
d) 停止位 - \texttt{i2c\_master\_stop()}

函数 \texttt{i2c\_master\_write\_byte()} 和 \texttt{i2c\_master\_write()} 都有额外的实参，规定主机是否应确认其有无接受到 ACK 位。

2. 通过调用 \texttt{i2c\_master\_cmd\_begin()} 来触发 I2C 控制器执行命令链接。一旦开始执行，就不能再修改命令链接。

3. 命令发送后，通过调用 \texttt{i2c\_cmd\_link\_delete()} 释放命令链接使用的资源。

**主机读取数据** 下面的示例展示如何为 I2C 主机构建命令链接，以便从从机读取 n 个字节。

在读取数据时，在上图的步骤 4 中，不是用 \texttt{i2c\_master\_write()}，而是用 \texttt{i2c\_master\_read\_byte()} 和 \texttt{i2c\_master\_read()} 填充命令链接。同样，在步骤 5 中配置最后一次的读取，以便主机不提供 ACK 位。

**指示写入或读取数据** 发送从机地址后，请参考图中第 3 步，主机可以写人或从从机读取数据。

主机实际执行的操作信息存储在从机地址的最低有效位中。

因此，为了将数据写入从机，主机发送的命令链接应包含地址 (ESP\_SLAVE\_ADDR << 1) | I2C\_MASTER\_WRITE，如下所示：

\begin{verbatim}
i2c\_master\_write\_byte(cmd, (ESP\_SLAVE\_ADDR << 1) | I2C\_MASTER\_WRITE, ACK\_EN);
\end{verbatim}

同理，指示从从机读取数据的命令链接如下所示：

\begin{verbatim}
i2c\_master\_write\_byte(cmd, (ESP\_SLAVE\_ADDR << 1) | I2C\_MASTER\_READ, ACK\_EN);
\end{verbatim}

**从机模式下通信** 安装 I2C 驱动程序后，ESP32 即可与其他 I2C 设备通信。

API 为从机提供以下功能：

- \texttt{i2c\_slave\_read\_buffer()}

当主机将数据写入从机时，从机将自动将其存储在接收缓存区中。从机应用程序可自行调用函数 \texttt{i2c\_slave\_read\_buffer()}。如果接收缓存区中没有数据，此函数还具有一个参数用于指定阻塞时间。这将允许从机应用程序在指定的超时设定内等待数据到达缓存区。

- \texttt{i2c\_slave\_write\_buffer()}

发送缓存区是用于存储从机要以 FIFO 顺序发送给主机的所有数据。在主机请求接收前，这些数据一直存储在发送缓存区。函数 \texttt{i2c\_slave\_write\_buffer()} 有一个参数，用于指定发送缓存区已满时的块时间。这将允许从机应用程序在指定的超时设定内等待发送缓存区中足够的可用空间。
在 peripherals/i2c 中可找到介绍如何使用这些功能的代码示例。

中断处理 安装驱动程序时，默认情况下会安装中断处理程序。

用户自定义配置 如本节末尾所述 配置驱动程序，函数i2c_param_config()在初始化 I2C 端口的驱动程序配置时，也会将几个 I2C 通信参数设置为 I2C 总线协议规范规定的默认值。其他一些相关参数已在 I2C 控制器的寄存器中预先配置。

通过调用下表中提供的专用函数，可以将所有这些参数更改为用户自定义值。请注意，时序值是在 APB 时钟周期中定义。APB 的频率在I2C_APB_CLK_FREQ 中指定。

### 表 4: 其他可配置的 I2C 通信参数

<table>
<thead>
<tr>
<th>要更改的参数</th>
<th>函数</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCL 脉冲周期的高电平和低电平</td>
<td>i2c_set_period()</td>
</tr>
<tr>
<td>在产生启动信号期间使用的 SCL 和 SDA 信号时序</td>
<td>i2c_set_start_timing()</td>
</tr>
<tr>
<td>在产生停止信号期间使用的 SCL 和 SDA 信号时序</td>
<td>i2c_set_stop_timing()</td>
</tr>
<tr>
<td>从机采样以及主机切换时，SCL 和 SDA 信号之间的时序关系</td>
<td>i2c_set_data_timing()</td>
</tr>
<tr>
<td>I2C 超时</td>
<td>i2c_set_timeout()</td>
</tr>
<tr>
<td>优先发送接收最高有效位 (LSB) 或最低有效位 (MSB)，可在 i2c_trans_mode_t 定义的模式中选择</td>
<td>i2c_set_data_mode()</td>
</tr>
</tbody>
</table>

上述每个函数都有一个_get_ 对应项来检查当前设置的值。例如，调用 i2c_get_timeout() 来检查 I2C 超时值。

要检查在驱动程序配置过程中设置的参数默认值，请参考文件 driver/i2c.c 并查找带有后缀 _DEFAULT 的定义。

通过函数i2c_set_pin() 可以为 SDA 和 SCL 信号选择不同的管脚并改变上拉配置。如果要修改已经输入的值，请使用函数i2c_param_config()。

### 备注: ESP32 的内部上拉电阻范围为几万欧姆，因此在大多数情况下，它们本身不足以用作 I2C 上拉电阻。建议用户使用阻值在 I2C 总线协议规范规定范围内的上拉电阻。计算阻值的具体方法，可参考 TI 应用说明

错误处理 大多数 I2C 驱动程序的函数在成功完成时会返回 ESP_OK，或在失败时会返回特定的错误代码。实时检查返回的值并进行错误处理是一种好习惯。驱动程序也会打印日志消息，其中包含错误说明，例如检查输入配置的正确性。有关详细信息，请参考文件 driver/i2c.c 并用后缀 _ERR_STR 查找定义。

使用专用中断来捕获通信故障。例如，如果主机将数据发送回主机耗费太长时间，会触发 I2C.TIME_OUT_INT 中断。详细信息请参考中断处理。

如果出现通信失败，可以分别为发送和接收缓存区调用 i2c_reset_tx_fifo() 和 i2c_reset_rx_fifo() 来重置内部硬件缓存区。

### 删除驱动程序 当使用 i2c_driver_install() 建立 I2C 通信，一段时间后不再需要 I2C 通信时，可以通过调用 i2c_driver_delete() 来移除驱动程序以释放分配的资源。

由于函数i2c_driver_delete() 无法保证线程安全性，请在调用该函数移除驱动程序前务必确保所有的线程都已停止使用驱动程序。

### 应用示例

I2C 主机和从机示例： peripherals/i2c。

Espressif Systems 1038 Release v5.1-dev-2066-g7869f4e151 Submit Document Feedback
## API 参考

### Header File

- components/driver/include/driver/i2c.h

### Functions

**esp_err_t i2c_driver_install**

```c
i2c_driver_install(i2c_port_t i2c_num, i2c_mode_t mode, size_t slv_rx_buf_len, size_t slv_tx_buf_len, int intr_alloc_flags)
```

Install an I2C driver.

**备注:** Not all Espressif chips can support slave mode (e.g. ESP32C2)

**备注:** In master mode, if the cache is likely to be disabled (such as write flash) and the slave is time-sensitive, ESP_INTR_FLAG_IRAM is suggested to be used. In this case, please use the memory allocated from internal RAM in i2c read and write function, because we can not access the psram (if psram is enabled) in interrupt handle function when cache is disabled.

**参数**

- **i2c_num** – I2C port number
- **mode** – I2C mode (either master or slave).
- **slv_rx_buf_len** – Receiving buffer size. Only slave mode will use this value, it is ignored in master mode.
- **slv_tx_buf_len** – Sending buffer size. Only slave mode will use this value, it is ignored in master mode.
- **intr_alloc_flags** – Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

**返回**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver installation error

**esp_err_t i2c_driver_delete**

```c
i2c_driver_delete(i2c_port_t i2c_num)
```

Delete I2C driver.

**备注:** This function does not guarantee thread safety. Please make sure that no thread will continuously hold semaphores before calling the delete function.

**参数**

- **i2c_num** – I2C port to delete

**返回**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**esp_err_t i2c_param_config**

```c
i2c_param_config(i2c_port_t i2c_num, const i2c_config_t *i2c_conf)
```

Configure an I2C bus with the given configuration.

**参数**

- **i2c_num** – I2C port to configure
- **i2c_conf** – Pointer to the I2C configuration

**返回**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
**esp_err_t i2c_reset_tx_fifo (i2c_port_t i2c_num)**
reset I2C tx hardware fifo

参数 i2c_num – I2C port number

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**esp_err_t i2c_reset_rx_fifo (i2c_port_t i2c_num)**
reset I2C rx fifo

参数 i2c_num – I2C port number

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**esp_err_t i2c_set_pin (i2c_port_t i2c_num, int sda_io_num, int scl_io_num, bool sda_pullup_en, bool scl_pullup_en, i2c_mode_t mode)**
Configure GPIO pins for I2C SCK and SDA signals.

参数
- i2c_num – I2C port number
- sda_io_num – GPIO number for I2C SDA signal
- scl_io_num – GPIO number for I2C SCL signal
- sda_pullup_en – Enable the internal pullup for SDA pin
- scl_pullup_en – Enable the internal pullup for SCL pin
- mode – I2C mode

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**esp_err_t i2c_master_write_to_device (i2c_port_t i2c_num, uint8_t device_address, const uint8_t *write_buffer, size_t write_size, TickType_t ticks_to_wait)**
Perform a write to a device connected to a particular I2C port. This function is a wrapper to i2c_master_start(), i2c_master_write(), i2c_master_read(), etc… It shall only be called in I2C master mode.

参数
- i2c_num – I2C port number to perform the transfer on
- device_address – I2C device’s 7-bit address
- write_buffer – Bytes to send on the bus
- write_size – Size, in bytes, of the write buffer
- ticks_to_wait – Maximum ticks to wait before issuing a timeout.

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Sending command error, slave hasn’t ACK the transfer.
- ESP_ERR_INVALID_STATE I2C driver not installed or not in master mode.
- ESP_ERR_TIMEOUT Operation timeout because the bus is busy.

**esp_err_t i2c_master_read_from_device (i2c_port_t i2c_num, uint8_t device_address, uint8_t *read_buffer, size_t read_size, TickType_t ticks_to_wait)**
Perform a read to a device connected to a particular I2C port. This function is a wrapper to i2c_master_start(), i2c_master_write(), i2c_master_read(), etc… It shall only be called in I2C master mode.

参数
- i2c_num – I2C port number to perform the transfer on
- device_address – I2C device’s 7-bit address
- read_buffer – Buffer to store the bytes received on the bus
- read_size – Size, in bytes, of the read buffer
• **ticks_to_wait** – Maximum ticks to wait before issuing a timeout.

### esp_err_t i2c_master_write_read_device

```
esp_err_t i2c_master_write_read_device(i2c_port_t i2c_num, uint8_t device_address, const uint8_t *write_buffer, size_t write_size, uint8_t *read_buffer, size_t read_size, TickType_t ticks_to_wait)
```

Perform a write followed by a read to a device on the I2C bus. A repeated start signal is used between the write and read, thus, the bus is not released until the two transactions are finished. This function is a wrapper to `i2c_master_start()`, `i2c_master_write()`, `i2c_master_read()`, etc. It shall only be called in I2C master mode.

### Parameters

- **i2c_num** – I2C port number to perform the transfer on
- **device_address** – I2C device’s 7-bit address
- **write_buffer** – Bytes to send on the bus
- **write_size** – Size, in bytes, of the write buffer
- **read_buffer** – Buffer to store the bytes received on the bus
- **read_size** – Size, in bytes, of the read buffer
- **ticks_to_wait** – Maximum ticks to wait before issuing a timeout.

### Return Values

- ESP_OK – Success
- ESP_ERR_INVALID_ARG – Parameter error
- ESP_FAIL – Sending command error, slave hasn’t ACK the transfer.
- ESP_ERR_INVALID_STATE – I2C driver not installed or not in master mode.
- ESP_ERR_TIMEOUT – Operation timeout because the bus is busy.

### i2c_cmd_handle_t i2c_cmd_link_create_static

Create and initialize an I2C commands list with a given buffer. All the allocations for data or signals (START, STOP, ACK, …) will be performed within this buffer. This buffer must be valid during the whole transaction. After finishing the I2C transactions, it is required to call `i2c_cmd_link_delete_static()`.

#### Parameters

- **buffer** – Buffer to use for commands allocations
- **size** – Size in bytes of the buffer

#### Return Value

Handle to the I2C command link or NULL if the buffer provided is too small, please use `I2C_LINK_RECOMMENDED_SIZE` macro to get the recommended size for the buffer.

### i2c_cmd_handle_t i2c_cmd_link_create

Create and initialize an I2C commands list with a given buffer. After finishing the I2C transactions, it is required to call `i2c_cmd_link_delete()` to release and return the resources. The required bytes will be dynamically allocated.

#### Parameters

- **cmd_handle** – I2C commands list allocated statically. This handle should be created thanks to `i2c_cmd_link_create_static()` function

---

**Note:** It is highly advised not to allocate this buffer on the stack. The size of the data used underneat may increase in the future, resulting in a possible stack overflow as the macro `I2C_LINK_RECOMMENDED_SIZE` would also return a bigger value. A better option is to use a buffer allocated statically or dynamically (with `malloc`).
void i2c_cmd_link_delete (i2c_cmd_handle_t cmd_handle)

Free the I2C commands list.

参数 cmd_handle - I2C commands list. This handle should be created thanks to i2c_cmd_link_create() function

esp_err_t i2c_master_start (i2c_cmd_handle_t cmd_handle)

Queue a “START signal” to the given commands list. This function shall only be called in I2C master mode. Call i2c_master_cmd_begin() to send all the queued commands.

参数 cmd_handle - I2C commands list

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create cmd_handler is too small
- ESP_FAIL No more memory left on the heap

esp_err_t i2c_master_write_byte (i2c_cmd_handle_t cmd_handle, uint8_t data, bool ack_en)

Queue a “write byte” command to the commands list. A single byte will be sent on the I2C port. This function shall only be called in I2C master mode. Call i2c_master_cmd_begin() to send all queued commands.

参数
- cmd_handle - I2C commands list
- data - Byte to send on the port
- ack_en - Enable ACK signal

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create cmd_handler is too small
- ESP_FAIL No more memory left on the heap

esp_err_t i2c_master_write (i2c_cmd_handle_t cmd_handle, const uint8_t *data, size_t data_len, bool ack_en)

Queue a “write (multiple) bytes” command to the commands list. This function shall only be called in I2C master mode. Call i2c_master_cmd_begin() to send all queued commands.

参数
- cmd_handle - I2C commands list
- data - Bytes to send. This buffer shall remain valid until the transaction is finished. If the PSRAM is enabled and intr_flag is set to ESP_INTR_FLAG_IRAM, data should be allocated from internal RAM.
- data_len - Length, in bytes, of the data buffer
- ack_en - Enable ACK signal

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM The static buffer used to create cmd_handler is too small
- ESP_FAIL No more memory left on the heap

esp_err_t i2c_master_read_byte (i2c_cmd_handle_t cmd_handle, uint8_t *data, i2c_ack_type_t ack)

Queue a “read byte” command to the commands list. A single byte will be read on the I2C bus. This function shall only be called in I2C master mode. Call i2c_master_cmd_begin() to send all queued commands.

参数
- cmd_handle - I2C commands list
- data - Pointer where the received byte will be stored. This buffer shall remain valid until the transaction is finished.
- ack - ACK signal

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
Chapter 2. API

• ESP_ERR_NO_MEM The static buffer used to create cmd_handler is too small
• ESP_FAIL No more memory left on the heap

```c
esp_err_t i2c_master_read(i2c_cmd_handle_t cmd_handle, uint8_t *data, size_t data_len, i2c_ack_type_t ack)
```

Queue a “read (multiple) bytes” command to the commands list. Multiple bytes will be read on the I2C bus. This function shall only be called in I2C master mode. Call i2c_master_cmd_begin() to send all queued commands.

参数

• **cmd_handle** - I2C commands list
• **data** - Pointer where the received bytes will be stored. This buffer shall remain valid until the transaction is finished.
• **data_len** - Size, in bytes, of the data buffer
• **ack** - ACK signal

返回

• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error
• ESP_ERR_NO_MEM The static buffer used to create cmd_handler is too small
• ESP_FAIL No more memory left on the heap

```c
esp_err_t i2c_master_stop(i2c_cmd_handle_t cmd_handle)
```

Queue a “STOP signal” to the given commands list. This function shall only be called in I2C master mode. Call i2c_master_cmd_begin() to send all the queued commands.

参数 **cmd_handle** - I2C commands list

返回

• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error
• ESP_ERR_NO_MEM The static buffer used to create cmd_handler is too small
• ESP_FAIL No more memory left on the heap

```c
esp_err_t i2c_master_cmd_begin(i2c_port_t i2c_num, i2c_cmd_handle_t cmd_handle, TickType_t ticks_to_wait)
```

Send all the queued commands on the I2C bus, in master mode. The task will be blocked until all the commands have been sent out. The I2C port is protected by mutex, so this function is thread-safe. This function shall only be called in I2C master mode.

参数

• **i2c_num** - I2C port number
• **cmd_handle** - I2C commands list
• **ticks_to_wait** - Maximum ticks to wait before issuing a timeout.

返回

• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error
• ESP_FAIL Sending command error, slave hasn’t ACK the transfer.
• ESP_ERR_INVALID_STATE I2C driver not installed or not in master mode.
• ESP_ERR_TIMEOUT Operation timeout because the bus is busy.

```c
int i2c_slave_write_buffer(i2c_port_t i2c_num, const uint8_t *data, int size, TickType_t ticks_to_wait)
```

Write bytes to internal ringbuffer of the I2C slave data. When the TX fifo empty, the ISR will fill the hardware FIFO with the internal ringbuffer’s data.

备注：This function shall only be called in I2C slave mode.

参数

• **i2c_num** - I2C port number
• **data** - Bytes to write into internal buffer
• **size** - Size, in bytes, of data buffer
**Chapter 2. API**

### `i2c_slave_read_buffer`

```c
int i2c_slave_read_buffer(i2c_port_t i2c_num, uint8_t* data, size_t max_size, TickType_t ticks_to_wait)
```

Read bytes from I2C internal buffer. When the I2C bus receives data, the ISR will copy them from the hardware RX FIFO to the internal ringbuffer. Calling this function will then copy bytes from the internal ringbuffer to the data user buffer.

**备注:** This function shall only be called in I2C slave mode.

**参数**
- `i2c_num` – I2C port number
- `data` – Buffer to fill with ringbuffer’s bytes
- `max_size` – Maximum bytes to read
- `ticks_to_wait` – Maximum waiting ticks

**返回**
- ESP_FAIL(-1) Parameter error
- Others(>=0) The number of data bytes read from I2C slave buffer.

### `i2c_set_period`

```c
esp_err_t i2c_set_period(i2c_port_t i2c_num, int high_period, int low_period)
```

Set I2C master clock period.

**参数**
- `i2c_num` – I2C port number
- `high_period` – Clock cycle number during SCL is high level, high_period is a 14 bit value
- `low_period` – Clock cycle number during SCL is low level, low_period is a 14 bit value

**返回**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

### `i2c_get_period`

```c
esp_err_t i2c_get_period(i2c_port_t i2c_num, int* high_period, int* low_period)
```

Get I2C master clock period.

**参数**
- `i2c_num` – I2C port number
- `high_period` – pointer to get clock cycle number during SCL is high level, will get a 14 bit value
- `low_period` – pointer to get clock cycle number during SCL is low level, will get a 14 bit value

**返回**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

### `i2c_filter_enable`

```c
esp_err_t i2c_filter_enable(i2c_port_t i2c_num, uint8_t cyc_num)
```

Enable hardware filter on I2C bus. Sometimes the I2C bus is disturbed by high frequency noise (about 20ns), or the rising edge of the SCL clock is very slow, these may cause the master state machine to break. Enable hardware filter can filter out high frequency interference and make the master more stable.

**备注:** Enable filter will slow down the SCL clock.

**参数**
- `i2c_num` – I2C port number to filter
• cyc_num – the APB cycles need to be filtered (0<=cyc_num <=7). When the period of a pulse is less than cyc_num * APB_cycle, the I2C controller will ignore this pulse.

返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error

esp_err_t i2c_filter_disable(i2c_port_t i2c_num)
Disable filter on I2C bus.
参数 i2c_num – I2C port number
返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error

esp_err_t i2c_set_start_timing(i2c_port_t i2c_num, int setup_time, int hold_time)
set I2C master start signal timing
参数
• i2c_num – I2C port number
• setup_time – clock number between the falling-edge of SDA and rising-edge of SCL for start mark, it’s a 10-bit value.
• hold_time – clock number between the falling-edge of SDA and falling-edge of SCL for start mark, it’s a 10-bit value.
返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error

esp_err_t i2c_get_start_timing(i2c_port_t i2c_num, int *setup_time, int *hold_time)
get I2C master start signal timing
参数
• i2c_num – I2C port number
• setup_time – pointer to get setup time
• hold_time – pointer to get hold time
返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error

esp_err_t i2c_set_stop_timing(i2c_port_t i2c_num, int setup_time, int hold_time)
set I2C master stop signal timing
参数
• i2c_num – I2C port number
• setup_time – clock number between the rising-edge of SCL and the rising-edge of SDA, it’s a 10-bit value.
• hold_time – clock number after the STOP bit’s rising-edge, it’s a 14-bit value.
返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error

esp_err_t i2c_get_stop_timing(i2c_port_t i2c_num, int *setup_time, int *hold_time)
get I2C master stop signal timing
参数
• i2c_num – I2C port number
• setup_time – pointer to get setup time.
• hold_time – pointer to get hold time.
返回
• ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error
**Chapter 2. API**

### esp_err_t i2c_set_data_timing(i2c_port_t i2c_num, int sample_time, int hold_time)

**set I2C data signal timing**

**参数**
- `i2c_num` – I2C port number
- `sample_time` – clock number I2C used to sample data on SDA after the rising-edge of SCL, it’s a 10-bit value
- `hold_time` – clock number I2C used to hold the data after the falling-edge of SCL, it’s a 10-bit value

**返回**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

### esp_err_t i2c_get_data_timing(i2c_port_t i2c_num, int *sample_time, int *hold_time)

**get I2C data signal timing**

**参数**
- `i2c_num` – I2C port number
- `sample_time` – pointer to get sample time
- `hold_time` – pointer to get hold time

**返回**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

### esp_err_t i2c_set_timeout(i2c_port_t i2c_num, int timeout)

**set I2C timeout value**

**参数**
- `i2c_num` – I2C port number
- `timeout` – timeout value for I2C bus (unit: APB 80Mhz clock cycle)

**返回**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

### esp_err_t i2c_get_timeout(i2c_port_t i2c_num, int *timeout)

**get I2C timeout value**

**参数**
- `i2c_num` – I2C port number
- `timeout` – pointer to get timeout value

**返回**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

### esp_err_t i2c_set_data_mode(i2c_port_t i2c_num, i2c_trans_mode_t tx_trans_mode, i2c_trans_mode_t rx_trans_mode)

**set I2C data transfer mode**

**参数**
- `i2c_num` – I2C port number
- `tx_trans_mode` – I2C sending data mode
- `rx_trans_mode` – I2C receiving data mode

**返回**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

### esp_err_t i2c_get_data_mode(i2c_port_t i2c_num, i2c_trans_mode_t *tx_trans_mode, i2c_trans_mode_t *rx_trans_mode)

**get I2C data transfer mode**

**参数**
- `i2c_num` – I2C port number
Chapter 2. API 参考

- **tx_trans_mode** - pointer to I2C sending data mode
- **rx_trans_mode** - pointer to I2C receiving data mode

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**Structures**

struct `i2c_config_t`

I2C initialization parameters.

**Public Members**

- `i2c_mode_t mode`
  I2C mode

- `int sda_io_num`
  GPIO number for I2C sda signal

- `int scl_io_num`
  GPIO number for I2C scl signal

- `bool sda_pullup_en`
  Internal GPIO pull mode for I2C sda signal

- `bool scl_pullup_en`
  Internal GPIO pull mode for I2C scl signal

- `uint32_t clk_speed`
  I2C clock frequency for master mode, (no higher than 1MHz for now)

- `struct i2c_config_t::[anonymous]::[anonymous] master`
  I2C master config

- `uint8_t addr_10bit_en`
  I2C 10bit address mode enable for slave mode

- `uint16_t slave_addr`
  I2C address for slave mode

- `uint32_t maximum_speed`
  I2C expected clock speed from SCL.

- `struct i2c_config_t::[anonymous]::[anonymous] slave`
  I2C slave config

- `uint32_t clk_flags`
  Bitwise of I2C_SCLK_SRC_FLAG_**FOR_DFS** for clk source choice
Macros

I2C_APB_CLK_FREQ
I2C source clock is APB clock, 80MHz

I2C_SCLK_SRC_FLAG_FOR_NOMAL
Any one clock source that is available for the specified frequency may be chosen

I2C_SCLK_SRC_FLAG_AWARE_DFS
For REF tick clock, it won’t change with APB.

I2C_SCLK_SRC_FLAG_LIGHT_SLEEP
For light sleep mode.

I2C_INTERNAL_STRUCT_SIZE
Minimum size, in bytes, of the internal private structure used to describe I2C commands link.

I2C_LINK_RECOMMENDED_SIZE (TRANSACTIONS)
The following macro is used to determine the recommended size of the buffer to pass to i2c_cmd_link_create_static() function. It requires one parameter, TRANSACTIONS, describing the number of transactions intended to be performed on the I2C port. For example, if one wants to perform a read on an I2C device register, TRANSACTIONS must be at least 2, because the commands required are the following:

• write device register
• read register content

Signals such as “(repeated) start”, “stop”, “nack”, “ack” shall not be counted.

Type Definitions

typedef void *i2c_cmd_handle_t
I2C command handle

Header File

• components/hal/include/hal/i2c_types.h

Structures

struct i2c_hal_clk_config_t
Data structure for calculating I2C bus timing.

Public Members

uint16_t clkm_div
I2C core clock devider

uint16_t scl_low
I2C scl low period
uint16_t scl_high
I2C scl high period

uint16_t scl_wait_high
I2C scl wait_high period

uint16_t sda_hold
I2C scl low period

uint16_t sda_sample
I2C sda sample time

uint16_t setup
I2C start and stop condition setup period

uint16_t hold
I2C start and stop condition hold period

uint16_t tout
I2C bus timeout period

struct i2c_hal_timing_config_t
Timing configuration structure. Used for I2C reset internally.

**Public Members**

int high_period
high_period time

int low_period
low_period time

int wait_high_period
wait_high_period time

int restart_setup
restart setup

int start_hold
start hold time

int stop_setup
stop setup

int stop_hold
stop hold time
```c
int sda_sample
    high_period time

int sda_hold
    sda hold time

int timeout
    timeout value
```

**Type Definitions**

typedef `soc_periph_i2c_clk_src_t` \_i2c\_clock\_source\_t
I2C group clock source.

**Enumerations**

eenum \_i2c\_port\_t
    I2C port number, can be I2C\_NUM\_0 ~ (I2C\_NUM\_MAX-1).
    Values:

    enumerator \_I2C\_NUM\_0
        I2C port 0

    enumerator \_I2C\_NUM\_1
        I2C port 1

    enumerator \_I2C\_NUM\_MAX
        I2C port max

eenum \_i2c\_mode\_t
    Values:

    enumerator \_I2C\_MODE\_SLAVE
        I2C slave mode

    enumerator \_I2C\_MODE\_MASTER
        I2C master mode

    enumerator \_I2C\_MODE\_MAX

eenum \_i2c\_rw\_t
    Values:

    enumerator \_I2C\_MASTER\_WRITE
        I2C write data

    enumerator \_I2C\_MASTER\_READ
        I2C read data
enum i2c_trans_mode_t

Values:

enumerator I2C_DATA_MODE_MSB_FIRST
I2C data msb first

enumerator I2C_DATA_MODE_LSB_FIRST
I2C data lsb first

enumerator I2C_DATA_MODE_MAX

enum i2c_addr_mode_t

Values:

enumerator I2C_ADDR_BIT_7
I2C 7bit address for slave mode

enumerator I2C_ADDR_BIT_10
I2C 10bit address for slave mode

enumerator I2C_ADDR_BIT_MAX

enum i2c_ack_type_t

Values:

enumerator I2C_MASTER_ACK
I2C ack for each byte read

enumerator I2C_MASTER_NACK
I2C nack for each byte read

enumerator I2C_MASTER_LAST_NACK
I2C nack for the last byte

enumerator I2C_MASTER_ACK_MAX

2.6.9 Inter-IC Sound (I2S)

Introduction

I2S (Inter-IC Sound) is a serial, synchronous communication protocol that is usually used for transmitting audio data between two digital audio devices.

ESP32 contains two I2S peripheral(s). These peripherals can be configured to input and output sample data via the I2S driver.

An I2S bus that communicate in Standard or TDM mode consists of the following lines:

- **MCLK**: Master clock line. It’s an optional signal depends on slave side, mainly used for offering a reference clock to the I2S slave device.
- **BCLK**: Bit clock line. The bit clock for data line.
• **WS**: Word(Slot) select line. It is usually used to identify the vocal tract except PDM mode.
• **DIN/DOUT**: Serial data input/output line. (Data will loopback internally if din and dout are set to a same GPIO)

And for the I2S bus that communicate in PDM mode, the lines are:
• **CLK**: PDM clock line.
• **DIN/DOUT**: Serial data input/output line.

Each I2S controller has the following features that can be configured by the I2S driver:
• Operation as system master or slave
• Capable of acting as transmitter or receiver
• DMA controller that allows for streaming sample data without requiring the CPU to copy each data sample

Each controller can operate in simplex communication mode. Thus, the two controllers can be combined to establish full-duplex communication.

### I2S File Structure

Public headers that need to be included in the I2S application
• `i2s.h`: The header file of legacy I2S APIs (for apps using legacy driver).
• `i2s_std.h`: The header file that provides standard communication mode specific APIs (for apps using new driver with standard mode).
• `i2s_pdm.h`: The header file that provides PDM communication mode specific APIs (for apps using new driver with PDM mode).
• `i2s_tdm.h`: The header file that provides TDM communication mode specific APIs (for apps using new driver with TDM mode).

Public headers that have been included in the headers above
• `i2s_types_legacy.h`: The legacy public types that only used in the legacy driver.
• `i2s_types.h`: The header file that provides public types.
• `i2s_common.h`: The header file that provides common APIs for all communication modes.
Chapter 2. API

I2S Clock

Clock Source

- `i2s_clock_src_t::I2S_CLK_SRC_DEFAULT`: Default PLL clock.
- `i2s_clock_src_t::I2S_CLK_SRC_PLL_160M`: 160 MHz PLL clock.
- `i2s_clock_src_t::I2S_CLK_SRC_APLL`: Audio PLL clock, more precise than `I2S_CLK_SRC_PLL_160M` in high sample rate applications. Its frequency is configurable according to the sample rate, but if APLL has been occupied by emac or other channels already, the APLL frequency is not allowed to change, the driver will try to work under this APLL frequency, if this APLL frequency can’t meet the requirements of I2S, the clock configuration will fail.

Clock Terminology

- **sample rate**: The number of sampled data in one second per slot.
- **sclk**: Source clock frequency. It is the frequency of the clock source.
- **mclk**: Master clock frequency. bclk is generate from this clock, mclk is mostly needed in the case that requires the MCLK signal as a reference clock to synchronize BCLK and WS between I2S master role and slave role.
- **bclk**: Bit clock frequency. Every tick of this clock stands for one data bit on data pin. It means there will be 8/16/24/32 bclk ticks in one slot, because the number of bclk ticks in one slot is equal to the `i2s_std_slot_config_t::slot_bit_width`.
- **lrck / ws**: Left/Right clock or word select clock. For non-PDM mode, its frequency is equal to the sample rate.

**备注**: Normally mclk should be the multiple of sample rate and bclk at the same time. This field `i2s_std_clk_config_t::mclk_multiple` means the multiple of mclk to the sample rate. If slot_bit_width is set to `I2S_SLOT_BIT_WIDTH_24BIT`, to keep mclk a multiple to the bclk, `i2s_std_clk_config_t::mclk_multiple` should be set to `I2S_MCLK_MULTIPLE_384`, otherwise the ws will be inaccurate. But in the most other cases, `I2S_MCLK_MULTIPLE_256` should be enough.

I2S Communication Mode

### Overview of All Modes

<table>
<thead>
<tr>
<th>Target</th>
<th>Standard</th>
<th>PDM TX</th>
<th>PDM RX</th>
<th>TDM</th>
<th>ADC/DAC</th>
<th>LCD/Camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32</td>
<td>I2S 0/1</td>
<td>I2S 0</td>
<td>I2S 0</td>
<td>none</td>
<td>12S 0</td>
<td>12S 0</td>
</tr>
<tr>
<td>ESP32S2</td>
<td>I2S 0</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>12S 0</td>
</tr>
<tr>
<td>ESP32C3</td>
<td>I2S 0</td>
<td>I2S 0</td>
<td>none</td>
<td>I2S 0</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>ESP32C6</td>
<td>I2S 0</td>
<td>I2S 0</td>
<td>none</td>
<td>I2S 0</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>ESP32S3</td>
<td>I2S 0/1</td>
<td>I2S 0</td>
<td>I2S 0</td>
<td>I2S 0/1</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

**Standard Mode** Standard mode always has left and right two sound channels which are called ‘slots’. These slots can support 8/16/24/32 bits width sample data. And the communication format for the slots mainly includes these following formats:

- **Philips Format**: Data signal have one bit shift comparing to the WS(word select) signal. And the duty of WS signal is 50%.
**Chapter 2. API 参考**

### Standard Philips Timing Diagram

- **BCLK**
- **WS**
- **DIN / DOUT**

- **slot_bit_width**
- **data_bit_width**

- **MSB**
- **LSB**

- **Left Slot**
- **Right Slot**

**MSB Format:** Almost same as philips format, but its data have no shift.

### Standard MSB Timing Diagram

- **BCLK**
- **WS**
- **DIN / DOUT**

- **slot_bit_width**
- **data_bit_width**

- **MSB**
- **LSB**

- **Left Slot**
- **Right Slot**

**PCM Short Format:** Data have one bit shift and meanwhile WS signal becomes a pulse lasting one BCLK(Bit Clock) cycle.

### Standard PCM Timing Diagram

- **BCLK**
- **WS**
- **DIN / DOUT**

- **slot_bit_width**
- **data_bit_width**

- **MSB**
- **LSB**

- **Left Slot**
- **Right Slot**

**PDM Mode (TX)**

PDM(Pulse-density Modulation) mode for tx channel can convert PCM data into PDM format which always has left and right slots. PDM TX can only support 16 bits width sample data. PDM TX is only supported on I2S0, it needs at least a CLK pin for clock signal and a DOUT pin for data signal (i.e. WS and SD signal in the following figure, the BCK signal is an internal bit sampling clock, not needed between PDM devices). This mode allows user to configure the up-sampling parameters `i2s_pdm_tx_clk_config_t::up_sample_fp` and `i2s_pdm_tx_clk_config_t::up_sample_fs`. The up-sampling rate can be calculated by `up_sample_rate = fp / fs`, there are up-sampling modes in PDM TX:

- **Fixed Clock Frequency:** In this mode the up-sampling rate will change according to the sample rate. Setting `fp = 960` and `fs = sample_rate / 100`, then the clock frequency(Fpdm) on CLK pin will be fixed to `128 * 48 KHz = 6.144 MHz`, note that this frequency is not equal to the sample rate(Fpcm).
- **Fixed Up-sampling Rate:** In this mode the up-sampling rate is fixed to 2. Setting `fp = 960` and `fs = 480`, then the clock frequency(Fpdm) on CLK pin will be `128 * sample_rate`
PDM Mode (RX)  PDM (Pulse-density Modulation) mode for rx channel can receive PDM format data and convert the data into PCM format. PDM RX is only supported on I2S0, it can only support 16 bits width sample data. PDM RX needs at least a CLK pin for clock signal and a DIN pin for data signal. This mode allows user to configure the down-sampling parameter `i2s_pdm_rx_clk_config_t::dn_sample_mode`, there are two down-sampling modes in PDM RX:

- `i2s_pdm_dsr_t::I2S_PDM_DSR_8S`: In this mode, the clock frequency (Fpdm) on WS pin will be `sample_rate(Fpcm) * 64`.
- `i2s_pdm_dsr_t::I2S_PDM_DSR_16S`: In this mode, the clock frequency (Fpdm) on WS pin will be `sample_rate(Fpcm) * 128`.

LCD/Camera Mode  LCD/Camera mode are only supported on I2S0 over a parallel bus. For LCD mode, I2S0 should working at master tx mode. For camera mode, I2S0 should working at slave rx mode. These two modes are not implemented by I2S driver, please refer to LCD for LCD implementation. For more information, see ESP32 Technical Reference Manual > I2S Controller (I2S) > LCD Mode [PDF].

ADC/DAC Mode  ADC and DAC modes only exist on ESP32 and are only supported on I2S0. Actually, they are two sub-modes of LCD/Camera mode. I2S0 can be routed directly to the internal analog-to-digital converter (ADC) and digital-to-analog converter (DAC). In other words, ADC and DAC peripherals can read or write continuously via I2S0 DMA. As they are not an actual communication mode, the I2S driver does not implement them.

Functional Overview

The I2S driver offers following services:

Resources Management  There are three levels' resources in I2S driver:

- platform level: Resources of all I2S controllers in the current target.
- controller level: Resources in one I2S controller.
- channel level: Resources of tx or rx channel in one I2S controller.

The public APIs are all channel level APIs, the channel handle `i2s_chan_handle_t` can help user to manage the resources under a specific channel without considering the other two levels. The other two upper levels' resources are private and will be managed by the driver automatically. Users can call `i2s_new_channel()` to allocate a channel handle and call `i2s_del_channel()` to delete it.

Power Management  When the power management is enabled (i.e. `CONFIG_PM_ENABLE` is on), the system will adjust or stop the source clock of I2S before going into light sleep, thus potentially changing the I2S signals and leading to transmitting or receiving invalid data.

I2S driver can prevent the system from changing or stopping the source clock by acquiring a power management lock. When the source clock is generated from APB, the lock type will be set to `esp_pm_lock_type_t::ESP_PM_APB_FREQ_MAX` and when the source clock is APLL (if target support APLL), it will be set to `esp_pm_lock_type_t::ESP_PM_NO_LIGHT_SLEEP`. Whenever user is reading or writing via I2S (i.e. calling `i2s_channel_read()` or `i2s_channel_write()`), the driver will guarantee that the power management lock is acquired. Likewise, the driver releases the lock after reading or writing finished.
Finite-State Machine  There are three states for an I2S channel, they are registered, ready and running. Their relationship is shown in the following diagram:

The <mode> in the diagram can be replaced by corresponding I2S communication mode like std for standard two-slot mode, for other information of communication mode, please refer to I2S Communication Mode section.

Data Transport  The data transport of I2S peripheral, including sending and receiving, is realized by DMA. Before transporting data, please call i2s_channel_enable() to enable the specific channel. When the sent or received data reach the size of one DMA buffer, I2S_OUT_EOF or I2S_IN_SUC_EOF interrupt will be triggered. Note that the DMA buffer size is not equal to i2s_chan_config_t::dma_frame_num, one frame here means all the sampled data in one WS circle. Therefore, dma_buffer_size = dma_frame_num * slot_num * slot_bit_width / 8. For the transmit case, users can input the data by calling i2s_channel_write(). This function will help users to copy the data from the source buffer to the DMA tx buffer and wait for the transmission finished. Then it’ll repeat until the sent bytes reach the given size. For the receive case, the function i2s_channel_read() will wait for receiving the message queue which contains the DMA buffer address, it will help users to copy the data from DMA rx buffer to the destination buffer.

Both i2s_channel_write() and i2s_channel_read() are blocking functions, they will keep waiting until the whole source buffer are sent or the whole destination buffer loaded, unless they exceed the max blocking time, then the error code ESP_ERR_TIMEOUT will return in this case. To send or receive data asynchronously, callbacks can be registered by i2s_channel_register_event_callback(), users are able to access the DMA buffer directly in the callback function instead of transmitting or receiving by the two blocking functions. However, please be aware that it is an interrupt callback, don’t do complex logic, floating operation or call non-reentrant functions in the callback.

Configuration Setting  Users can initialize a channel by corresponding function (i.e. i2s_channel_init_std_mode(), i2s_channel_init_pdm_rx_mode(),
Chapter 2. API

i2s_channel_init_pdm_tx_mode() or i2s_channel_init_tdm_mode(), the channel will be initialized to the specific mode. If the configurations need to be updated after initialization, i2s_channel_disable() has to be called first to ensure the channel has stopped, and then calling corresponding reconfig functions, like i2s_channel_reconfig_std_slot(), i2s_channel_reconfig_std_clock(), i2s_channel_reconfig_std_gpio().

IRAM Safe By default, the I2S interrupt will be deferred when the Cache is disabled for reasons like writing/erasing Flash. Thus the EOF interrupt will not get executed in time, which is not expected in a real-time application.

There’s a Kconfig option CONFIG_I2S_ISR_IRAM_SAFE that will:

1. Enable the interrupt being serviced even when cache is disabled
2. Place driver object into DRAM (in case it’s linked to PSRAM by accident)

This will allow the interrupt to run while the cache is disabled but will come at the cost of increased IRAM consumption.

Thread Safety All the public I2S APIs are guaranteed to be thread safe by the driver, which means, user can call them from different RTOS tasks without protection by extra locks. Notice that I2S driver uses mutex lock to ensure the thread safety, thus these APIs are not allowed to be used in ISR.

Kconfig Options

- CONFIG_I2S_ISR_IRAM_SAFE controls whether the default ISR handler can work when cache is disabled, see IRAM Safe for more information.
- CONFIG_I2S_SUPPRESS_DEPRECATED_WARN controls whether to suppress the compiling warning message while using the legacy I2S driver.
- CONFIG_I2S_ENABLE_DEBUG_LOG is used to enabled the debug log output. Enable this option will increase the firmware binary size.

Application Example

The examples of the I2S driver can be found in the directory peripherals/i2s. Here are some simple usages of each mode:

Standard TX/RX Usage Different slot communication formats can be generated by following helper macros for standard mode. As described above, there are three formats in standard mode, their helper macros are:

- I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG
- I2S_STD_PCM_SLOT_DEFAULT_CONFIG
- I2S_STD_MSB_SLOT_DEFAULT_CONFIG

The clock config helper macro is:

- I2S_STD_CLK_DEFAULT_CONFIG

Please refer to Standard Mode for STD API information. And for more details, please refer to driver/include/driver/i2s_std.h.

STD TX Mode Take 16-bit data width for example, when the data in a uint16_t writing buffer are:

| data 0 | data 1 | data 2 | data 3 | data 4 | data 5 | data 6 | data 7 | ...
|--------|--------|--------|--------|--------|--------|--------|--------|-------|
| 0x0001 | 0x0002 | 0x0003 | 0x0004 | 0x0005 | 0x0006 | 0x0007 | 0x0008 | ...

Here is the table of the real data on the line with different i2s_std_slot_config_t::slot_mode and i2s_std_slot_config_t::slot_mask
Chapter 2. API 參考

### 2.1 API 參考

<table>
<thead>
<tr>
<th>data</th>
<th>slot</th>
<th>mode</th>
<th>slot</th>
<th>mask</th>
<th>ws</th>
<th>low</th>
<th>high</th>
<th>ws</th>
<th>low</th>
<th>high</th>
<th>ws</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bit</td>
<td>mono</td>
<td>left</td>
<td>0x0002</td>
<td>0x0000</td>
<td>0x0001</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0004</td>
<td>0x0000</td>
<td>0x0003</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>right</td>
<td>0x0000</td>
<td>0x0002</td>
<td>0x0000</td>
<td>0x0001</td>
<td>0x0000</td>
<td>0x0004</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0003</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>both</td>
<td>0x0000</td>
<td>0x0002</td>
<td>0x0000</td>
<td>0x0001</td>
<td>0x0000</td>
<td>0x0004</td>
<td>0x0000</td>
<td>0x0003</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stereo</td>
<td>mono</td>
<td>left</td>
<td>0x0001</td>
<td>0x0001</td>
<td>0x0003</td>
<td>0x0003</td>
<td>0x0005</td>
<td>0x0005</td>
<td>0x0007</td>
<td>0x0007</td>
<td>0x0008</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>right</td>
<td>0x0002</td>
<td>0x0002</td>
<td>0x0004</td>
<td>0x0004</td>
<td>0x0006</td>
<td>0x0006</td>
<td>0x0008</td>
<td>0x0008</td>
<td>0x0008</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

备注：It’s similar when the data is 32-bit width, but take care when using 8-bit and 24-bit data width. For 8-bit width, the written buffer should still use uint16_t (i.e. align with 2 bytes), and only the high 8 bits will be valid, the low 8 bits are dropped, and for 24-bit width, the buffer is supposed to use uint32_t (i.e. align with 4 bytes), and only the high 24 bits valid, the low 8 bits are dropped.

Another point is that, for the 8-bit and 16-bit mono mode, the real data on the line are swapped. To get the correct sequence, the writing buffer need to swap the data every two bytes.

```c
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t tx_handle;
/* Get the default channel configuration by helper macro. */
/* This helper macro is defined in 'i2s_common.h' and shared by all the i2s_-
 communication mode. */
/* It can help to specify the I2S role, and port id */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE-
MASTER);
/* Allocate a new tx channel and get the handle of this channel */
i2s_new_channel(&chan_cfg, &tx_handle, NULL);

/* Setting the configurations, the slot configuration and clock configuration can-
be generated by the macros */
/* These two helper macros is defined in 'i2s_std.h' which can only be used in STD-
mode. */
/* They can help to specify the slot and clock configurations for initialization-
or updating */
i2s_std_config_t std_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(48000),
    .slot_cfg = I2S_STD_MSB_SLOT_DEFAULT_CONFIG(32BIT, I2S_SLOT-
MODE_STEREO),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .din = I2S_GPIO_UNUSED,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
/* Initialize the channel */
i2s_channel_init_std_mode(tx_handle, &std_cfg);
/* Before write data, start the tx channel first */
i2s_channel_enable(tx_handle);
```

(下頁續接)
Chapter 2. API 参考

```c
i2s_channel_write(tx_handle, src_buf, bytes_to_write, bytes_written, ticks_to_wait);

/* If the configurations of slot or clock need to be updated, */
/* stop the channel first and then update it */
// i2s_channel_disable(tx_handle);
// std_cfg.slot_cfg.slot_mode = I2S_SLOT_MODE_MONO; // Default is stereo
// i2s_channel_reconfig_std_slot(tx_handle, &std_cfg.slot_cfg);
// std_cfg.clk_cfg.sample_rate_hz = 96000;
// i2s_channel_reconfig_std_clock(tx_handle, &std_cfg.clk_cfg);

/* Have to stop the channel before deleting it */
i2s_channel_disable(tx_handle);
/* If the handle is not needed any more, delete it to release the channel resources */
i2s_del_channel(tx_handle);
```

### STD RX Mode

Take 16-bit data width for example, when the data on the line are:

- **ws low**
- **ws high**
- **ws low**
- **ws high**
- **ws low**
- **ws high**
- **ws low**
- **ws high**

Here is the table of the data that received in the buffer with different `i2s_std_slot_config_t::slot_mode` and `i2s_std_slot_config_t::slot_mask`:

<table>
<thead>
<tr>
<th>Data width</th>
<th>Bit width</th>
<th>Slot mode</th>
<th>Slot mask</th>
<th>Data 0</th>
<th>Data 1</th>
<th>Data 2</th>
<th>Data 3</th>
<th>Data 4</th>
<th>Data 5</th>
<th>Data 6</th>
<th>Data 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bit</td>
<td>16 bit</td>
<td>mono</td>
<td>left</td>
<td>0x0001</td>
<td>0x0000</td>
<td>0x0005</td>
<td>0x0003</td>
<td>0x0009</td>
<td>0x0007</td>
<td>0x000d</td>
<td>0x000b</td>
</tr>
<tr>
<td></td>
<td></td>
<td>right</td>
<td>0x0002</td>
<td>0x0000</td>
<td>0x0006</td>
<td>0x0004</td>
<td>0x000a</td>
<td>0x0008</td>
<td>0x000c</td>
<td>0x000e</td>
<td>0x000c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stereo</td>
<td>any</td>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
<td>0x0008</td>
</tr>
</tbody>
</table>

备注：The receive case is a little bit complicated on ESP32. Firstly, when the data width are 8-bit or 24-bit, the received data will still align with two bytes or four bytes, which means the valid data are put in the high 8 bits in every two bytes and high 24 bits in every four bytes. For example, the received data will be 0x5A00 when the data on the line is 0x5A in 8-bit width, and receive 0x0000 5A00 if the data 0x00 005A on the line. Secondly, for 8-bit and 16-bit mono case, the data in buffer are swapped every two data, they may need to be swapped back manually to get the correct order.

```c
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t rx_handle;
/* Get the default channel configuration by helper macro. */
/* This helper macro is defined in 'i2s_common.h' and shared by all the i2s communication mode. */
/* It can help to specify the I2S role, and port id */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_MASTER);
/* Allocate a new rx channel and get the handle of this channel */
i2s_new_channel(&chan_cfg, NULL, &rx_handle);

/* Setting the configurations, the slot configuration and clock configuration can be generated by the macros */
/* These two helper macros is defined in 'i2s_std.h' which can only be used in STD mode. */
```

(下頁続き)
* They can help to specify the slot and clock configurations for initialization.*

```c
i2s_std_config_t std_cfg = {
  .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(48000),
  .slot_cfg = I2S_STD_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_MODE_STEREO),
  .gpio_cfg = {
    .mclk = I2S_GPIO_UNUSED,
    .bclk = GPIO_NUM_4,
    .ws = GPIO_NUM_5,
    .dout = I2S_GPIO_UNUSED,
    .din = GPIO_NUM_19,
    .invert_flags = {
      .mclk_inv = false,
      .bclk_inv = false,
      .ws_inv = false,
    },
  },
};
/* Initialize the channel */
i2s_channel_init_std_mode(rx_handle, &std_cfg);
/* Before read data, start the rx channel first */
i2s_channel_enable(rx_handle);
i2s_channel_read(rx_handle, desc_buf, bytes_to_read, bytes_read, ticks_to_wait);
/* Have to stop the channel before deleting it */
i2s_channel_disable(rx_handle);
/* If the handle is not needed any more, delete it to release the channel resources */
i2s_del_channel(rx_handle);
```

**PDM TX usage**

For PDM mode in tx channel, the slot configuration helper macro is:

- `I2S_PDM_TX_SLOT_DEFAULT_CONFIG`

The clock configuration helper macro is:

- `I2S_PDM_TX_CLK_DEFAULT_CONFIG`

Please refer to [PDM Mode](#) for PDM TX API information. And for more details, please refer to `driver/include/driver/i2s_pdm.h`.

The PDM data width is fixed to 16-bit, when the data in a `int16_t` writing buffer are:

| data 0 | data 1 | data 2 | data 3 | data 4 | data 5 | data 6 | data 7 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
<td>0x0008</td>
<td>...</td>
</tr>
</tbody>
</table>

Here is the table of the real data on the line with different `i2s_pdm_tx_slot_config_t::slot_mode` and `i2s_pdm_tx_slot_config_t::slot_mask` (The PDM format on the line is transferred to PCM format for better comprehension).

<table>
<thead>
<tr>
<th>slot mode</th>
<th>slot mask</th>
<th>left</th>
<th>right</th>
<th>left</th>
<th>right</th>
<th>left</th>
<th>right</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>mono</td>
<td>left</td>
<td>0x0001</td>
<td>0x0000</td>
<td>0x0002</td>
<td>0x0000</td>
<td>0x0000</td>
<td>0x0004</td>
<td>0x0000</td>
<td>0x0000</td>
</tr>
<tr>
<td></td>
<td>right</td>
<td>0x0000</td>
<td>0x0001</td>
<td>0x0000</td>
<td>0x0002</td>
<td>0x0000</td>
<td>0x0003</td>
<td>0x0000</td>
<td>0x0004</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td>0x0001</td>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0004</td>
</tr>
<tr>
<td>stereo</td>
<td>left</td>
<td>0x0001</td>
<td>0x0001</td>
<td>0x0003</td>
<td>0x0003</td>
<td>0x0005</td>
<td>0x0007</td>
<td>0x0007</td>
<td>0x0007</td>
</tr>
<tr>
<td></td>
<td>right</td>
<td>0x0002</td>
<td>0x0002</td>
<td>0x0004</td>
<td>0x0004</td>
<td>0x0006</td>
<td>0x0006</td>
<td>0x0008</td>
<td>0x0008</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0004</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0005</td>
<td>0x0007</td>
<td>0x0008</td>
</tr>
</tbody>
</table>
# include "driver/i2s_pdm.h"
# include "driver/gpio.h"

/* Allocate an I2S tx channel */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_0, I2S_ROLE_MASTER);
i2s_new_channel(&chan_cfg, &tx_handle, NULL);

/* Init the channel into PDM TX mode */
i2s_pdm_tx_config_t pdm_tx_cfg = {
    .clk_cfg = I2S_PDM_TX_CLK_DEFAULT_CONFIG(36000),
    .slot_cfg = I2S_PDM_TX_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_MODE_MONO),
    .gpio_cfg = {
        .clk = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .invert_flags = {
            .clk_inv = false,
        },
    },
};
i2s_channel_init_pdm_tx_mode(tx_handle, &pdm_tx_cfg);

PDM RX usage For PDM mode in RX channel, the slot configuration helper macro is:

• I2S_PDM_RX_SLOT_DEFAULT_CONFIG

The clock configuration helper macro is:

• I2S_PDM_RX_CLK_DEFAULT_CONFIG

Please refer to PDM Mode for PDM RX API information. And for more details, please refer to driver/include/driver/i2s_pdm.h.

The PDM data width is fixed to 16-bit, when the data on the line (The PDM format on the line is transferred to PCM format for easier comprehension) are:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
<td>0x0008</td>
</tr>
</tbody>
</table>

Here is the table of the data that received in a `int16_t` buffer with different i2s_pdm_rx_slot_config_t::slot_mode and i2s_pdm_rx_slot_config_t::slot_mask

<table>
<thead>
<tr>
<th>slot mode</th>
<th>slot mask</th>
<th>data 0</th>
<th>data 1</th>
<th>data 2</th>
<th>data 3</th>
<th>data 4</th>
<th>data 5</th>
<th>data 6</th>
<th>data 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>mono</td>
<td>left</td>
<td>0x0001</td>
<td>0x0003</td>
<td>0x0005</td>
<td>0x0007</td>
<td>0x0009</td>
<td>0x000b</td>
<td>0x000d</td>
<td>0x000f</td>
</tr>
<tr>
<td></td>
<td>right</td>
<td>0x0002</td>
<td>0x0004</td>
<td>0x0006</td>
<td>0x0008</td>
<td>0x000a</td>
<td>0x000c</td>
<td>0x000e</td>
<td>0x0010</td>
</tr>
<tr>
<td>stereo</td>
<td>both</td>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
<td>0x0008</td>
</tr>
</tbody>
</table>

#include "driver/i2s_pdm.h"
#include "driver/gpio.h"

i2s_chan_handle_t rx_handle;

/* Allocate an I2S rx channel */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_0, I2S_ROLE_MASTER);
i2s_new_channel(&chan_cfg, NULL, &rx_handle);
/* Init the channel into PDM RX mode */
i2s_pdm_rx_config_t pdm_rx_cfg = {
    .clk_cfg = I2S_PDM_RX_CLK_DEFAULT_CONFIG(36000),
    .slot_cfg = I2S_PDM_RX_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_MODE_MONO),
    .gpio_cfg = {
        .clk = GPIO_NUM_5,
        .din = GPIO_NUM_19,
        .invert_flags = {
            .clk_inv = false,
        },
    },
};
i2s_channel_init_pdm_rx_mode(rx_handle, &pdm_rx_cfg);

Full-duplex  Full-duplex mode will register tx and rx channel in an I2S port at the same time, and they will share the BCLK and WS signal. Currently STD and TDM communication mode are able to adopt full-duplex mode in following way, but PDM full-duplex is not supported because PDM TX and RX clock are not same.

Note that one handle can only stand for one channel, the slot and clock configurations for both tx and rx channel should be set one by one.

Here is an example of how to allocate a pair of full-duplex channels:

```c
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t tx_handle;
i2sChan_handle_t rx_handle;

/* Allocate a pair of I2S channel */
i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_MASTER);
/* Allocate for tx and rx channel at the same time, then they will work in full-duplex mode */
i2s_new_channel(&chan_cfg, &tx_handle, &rx_handle);

/* Set the configurations for BOTH TWO channels, since tx and rx channel have to be same in full-duplex mode */
i2s_std_config_t std_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(32000),
    .slot_cfg = I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_MODE_STEREO),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .din = GPIO_NUM_19,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
i2s_init_channel(tx_handle, &std_cfg);
```
Chapter 2. API

Simplex Mode To allocate a channel handle in simplex mode, `i2s_new_channel()` should be called for each channel. The clock and gpio pins of TX/RX channel on ESP32 are not separate, therefore TX and RX channel can’t coexist on a same I2S port in simplex mode.

```c
#include "driver/i2s_std.h"
#include "driver/gpio.h"

i2s_chan_handle_t tx_handle;
i2s_chan_handle_t rx_handle;

i2s_chan_config_t chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_MASTER);
i2s_new_channel(chan_cfg, &tx_handle, NULL);
i2s_std_config_t std_tx_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(48000),
    .slot_cfg = I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_MODE_STEREO),
    .gpio_cfg = {
        .mclk = GPIO_NUM_0,
        .bclk = GPIO_NUM_4,
        .ws = GPIO_NUM_5,
        .dout = GPIO_NUM_18,
        .din = I2S_GPIO_UNUSED,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
/* Initialize the channel */
i2s_channel_init_std_mode(tx_handle, &std_tx_cfg);
i2s_channel_enable(tx_handle);

/* rx channel will be registered on another I2S, if no other available I2S unit... */
i2s_new_channel(chan_cfg, NULL, &rx_handle);
i2s_std_config_t std_rx_cfg = {
    .clk_cfg = I2S_STD_CLK_DEFAULT_CONFIG(16000),
    .slot_cfg = I2S_STD_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_MODE_STEREO),
    .gpio_cfg = {
        .mclk = I2S_GPIO_UNUSED,
        .bclk = GPIO_NUM_6,
        .ws = GPIO_NUM_7,
        .dout = I2S_GPIO_UNUSED,
        .din = GPIO_NUM_19,
        .invert_flags = {
            .mclk_inv = false,
            .bclk_inv = false,
            .ws_inv = false,
        },
    },
};
Application Notes

How to Prevent Data Lost  For the applications that need a high frequency sample rate, sometimes the massive throughput of receiving data may cause data lost. Users can receive data lost event by registering isr callback function to receive event queue:

```c
static IRAM_ATTR bool i2s_rx_queue_overflow_callback(i2s_chan_handle_t *handle, i2s_event_data_t *event, void *user_ctx)
{
    // handle rx queue overflow event ...
    return false;
}
```

Please follow these steps to prevent data lost:

1. Determine the interrupt interval. Generally, when data lost happened, the interval should be the bigger the better, it can help to reduce the interrupt times, i.e., dma_frame_num should be as big as possible while the DMA buffer size won’t exceed its maximum value 4092. The relationships are:

   \[
   \text{interrupt interval (unit: sec)} = \frac{\text{dma_frame_num}}{\text{sample rate}}
   \]

   \[
   \text{dma_buffer_size} = \frac{\text{dma_frame_num} \times \text{slot_num} \times \text{data_bit_width}}{8} < 4092
   \]

2. Determine the dma_desc_num. The dma_desc_num is decided by the max time of `i2s_channel_read` polling cycle, all the received data are supposed to be stored between two `i2s_channel_read`. This cycle can be measured by a timer or an outputting gpio signal. The relationship is:

   \[
   \text{dma_desc_num} > \frac{\text{polling_cycle}}{\text{interrupt interval}}
   \]

3. Determine the receiving buffer size. The receiving buffer that offered by user in `i2s_channel_read` should be able to take all the data in all dma buffers, that means it should be bigger than the total size of all the dma buffers:

   \[
   \text{recv_buffer_size} > \text{dma_desc_num} \times \text{dma_buffer_size}
   \]

For example, if there is an I2S application, and the known values are:

- `sample_rate = 144000 Hz`
- `data_bit_width = 32 bits`
- `slot_num = 2`
- `polling_cycle = 10ms`

Then the parameters `dma_frame_num`, `dma_desc_num` and `recv_buf_size` can be calculated according to the given known values:

\[
\text{dma_frame_num} \times \text{slot_num} \times \text{data_bit_width} / 8 = \text{dma_buffer_size} < 4092
\]

\[
\text{dma_frame_num} < 511
\]
interrupt_interval = dma_frame_num / sample_rate = 511 / 144000 = 0.003549 s = 3.549 ms
dma_desc_num > polling_cycle / interrupt_interval = cell(10 / 3.549) = cell(2.818)...
recv_buffer_size > dma_desc_num * dma_buffer_size = 3 * 4092 = 12276 bytes

API Reference

Standard Mode

Header File

- components/driver/include/driver/i2s_std.h

Functions

esp_err_t i2s_channel_init_std_mode(i2s_chan_handle_t handle, const i2s_std_config_t *std_cfg)

Initialize i2s channel to standard mode.

备注: Only allowed to be called when the channel state is REGISTERED, (i.e., channel has been allocated, but not initialized) and the state will be updated to READY if initialization success, otherwise the state will return to REGISTERED.

参数

- handle [in] I2S channel handler
- std_cfg [in] Configurations for standard mode, including clock, slot and gpio The clock configuration can be generated by the helper macro
  I2S_STD_CLK_DEFAULT_CONFIG The slot configuration can be generated by the helper macro
  I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG, I2S_STD_PCM_SLOT_DEFAULT_CONFIG or
  I2S_STD_MSB_SLOT_DEFAULT_CONFIG

返回

- ESP_OK Initialize successfully
- ESP_ERR_NO_MEM No memory for storing the channel information
- ESP_ERR_INVALID_ARG NULL pointer or invalid configuration
- ESP_ERR_INVALID_STATE This channel is not registered

esp_err_t i2s_channel_reconfig_std_clock(i2s_chan_handle_t handle, const i2s_std_clk_config_t *clk_cfg)

Reconfigure the I2S clock for standard mode.

备注: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won’t change the state. ‘i2s_channel_disable’ should be called before calling this function if i2s has started.

备注: The input channel handle has to be initialized to standard mode, i.e., ‘i2s_channel_init_std_mode’ has been called before reconfiguring

参数

- handle [in] I2S channel handler
- clk_cfg [in] Standard mode clock configuration, can be generated by
  I2S_STD_CLK_DEFAULT_CONFIG
Chapter 2. API

**ESP_OK** Set clock successfully  
**ESP_ERR_INVALID_ARG** NULL pointer, invalid configuration or not standard mode  
**ESP_ERR_INVALID_STATE** This channel is not initialized or not stopped

```c
esp_err_t i2s_channel_reconfig_std_slot (i2s_chan_handle_t handle, const i2s_std_slot_config_t *slot_cfg)
```

Reconfigure the I2S slot for standard mode.

**备注**: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won’t change the state. ‘i2s_channel_disable’ should be called before calling this function if i2s has started.

**备注**: The input channel handle has to be initialized to standard mode, i.e., ‘i2s_channel_init_std_mode’ has been called before reconfiguring

---

**参数**

- **handle**  [in] I2S channel handler
- **slot_cfg**  [in] Standard mode slot configuration, can be generated by `I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG`, `I2S_STD_PCM_SLOT_DEFAULT_CONFIG` and `I2S_STD_MSB_SLOT_DEFAULT_CONFIG`.

**返回**

- **ESP_OK** Set clock successfully
- **ESP_ERR_NO_MEM** No memory for DMA buffer
- **ESP_ERR_INVALID_ARG** NULL pointer, invalid configuration or not standard mode
- **ESP_ERR_INVALID_STATE** This channel is not initialized or not stopped

```c
esp_err_t i2s_channel_reconfig_std_gpio (i2s_chan_handle_t handle, const i2s_std_gpio_config_t *gpio_cfg)
```

Reconfigure the I2S gpio for standard mode.

**备注**: Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started this function won’t change the state. ‘i2s_channel_disable’ should be called before calling this function if i2s has started.

**备注**: The input channel handle has to be initialized to standard mode, i.e., ‘i2s_channel_init_std_mode’ has been called before reconfiguring

---

**参数**

- **handle**  [in] I2S channel handler
- **gpio_cfg**  [in] Standard mode gpio configuration, specified by user

**返回**

- **ESP_OK** Set clock successfully
- **ESP_ERR_NO_MEM** No memory for DMA buffer
- **ESP_ERR_INVALID_ARG** NULL pointer, invalid configuration or not standard mode
- **ESP_ERR_INVALID_STATE** This channel is not initialized or not stopped

**Structures**

```c
struct i2s_std_slot_config_t
```

I2S slot configuration for standard mode.


**Public Members**

```c
i2s_data_bit_width_t data_bit_width
```
I2S sample data bit width (valid data bits per sample)

```c
i2s_slot_bit_width_t slot_bit_width
```
I2S slot bit width (total bits per slot)

```c
i2s_slot_mode_t slot_mode
```
Set mono or stereo mode with I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO In TX direction, mono means the written buffer contains only one slot data and stereo means the written buffer contains both left and right data

```c
i2s_std_slot_mask_t slot_mask
```
Select the left, right or both slot

```c
uint32_t ws_width
```
WS signal width (i.e. the number of belk ticks that ws signal is high)

```c
bool ws_pol
```
WS signal polarity, set true to enable high lever first

```c
bool bit_shift
```
Set to enable bit shift in Philips mode

```c
bool msb_right
```
Set to place right channel data at the MSB in the FIFO

```c
struct i2s_std_clk_config_t
```
I2S clock configuration for standard mode.

**Public Members**

```c
uint32_t sample_rate_hz
```
I2S sample rate

```c
i2s_clock_src_t clk_src
```
Choose clock source

```c
i2s_mclk_multiple_t mclk_multiple
```
The multiple of mclk to the sample rate Default is 256 in the helper macro, it can satisfy most of cases, but please set this field a multiple of ‘3’ (like 384) when using 24-bit data width, otherwise the sample rate might be inaccurate

```c
struct i2s_std_gpio_config_t
```
I2S standard mode GPIO pins configuration.
Chapter 2. API

Public Members

gpio_num_t mclk
   MCK pin, output

gpio_num_t bclk
   BCK pin, input in slave role, output in master role

gpio_num_t ws
   WS pin, input in slave role, output in master role

gpio_num_t dout
   DATA pin, output

gpio_num_t din
   DATA pin, input

uint32_t mclk_inv
   Set 1 to invert the mclk output

uint32_t bclk_inv
   Set 1 to invert the bclk input/output

uint32_t ws_inv
   Set 1 to invert the ws input/output

struct i2s_std_gpio_config_t::{anonymous} invert_flags
   GPIO pin invert flags

struct i2s_std_config_t
   I2S standard mode major configuration that including clock/slot/gpio configuration.

Public Members

i2s_std_clk_config_t clk_cfg
   Standard mode clock configuration, can be generated by macro I2S_STD_CLK_DEFAULT_CONFIG

i2s_std_slot_config_t slot_cfg
   Standard mode slot configuration, can be generated by macros I2S_STD_[mode]_SLOT_DEFAULT_CONFIG, [mode] can be replaced with PHILIPS/MSB/PCM

i2s_std_gpio_config_t gpio_cfg
   Standard mode gpio configuration, specified by user

Macros

I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG (bits_per_sample, mono_or_stereo)
   Philips format in 2 slots.
   This file is specified for I2S standard communication mode Features:
• Philips/MSB/PCM are supported in standard mode
• Fixed to 2 slots

参数
• \texttt{bits\_per\_sample} - i2s data bit width
• \texttt{mono\_or\_stereo} - I2S SLOT_MODE_MONO or I2S SLOT_MODE_STEREO

\texttt{I2S\_STD\_PCM\_SLOT\_DEFAULT\_CONFIG}(\texttt{bits\_per\_sample, mono\_or\_stereo})

PCM(short) format in 2 slots.

备注: PCM(long) is same as philips in 2 slots

参数
• \texttt{bits\_per\_sample} - i2s data bit width
• \texttt{mono\_or\_stereo} - I2S SLOT_MODE_MONO or I2S SLOT_MODE_STEREO

\texttt{I2S\_STD\_MSB\_SLOT\_DEFAULT\_CONFIG}(\texttt{bits\_per\_sample, mono\_or\_stereo})

MSB format in 2 slots.

参数
• \texttt{bits\_per\_sample} - i2s data bit width
• \texttt{mono\_or\_stereo} - I2S SLOT_MODE_MONO or I2S SLOT_MODE_STEREO

\texttt{I2S\_STD\_CLK\_DEFAULT\_CONFIG}(\texttt{rate})

i2s default standard clock configuration

备注: Please set the mclk\_multiple to I2S\_MCLK\_MULTIPLE\_384 while using 24 bits data width Otherwise the sample rate might be imprecise since the bclk division is not a integer

参数
• \texttt{rate} - sample rate

PDM Mode

Header File
• \texttt{components/driver/include/driver/i2s\_pdm.h}

Functions
\texttt{esp\_err\_t i2s\_channel\_init\_pdm\_rx\_mode}(i2s\_chan\_handle\_t handle, const i2s\_pdm\_rx\_config\_t *pdm\_rx\_cfg)

Initialize i2s channel to PDM RX mode.

备注: Only allowed to be called when the channel state is REGISTERED, (i.e., channel has been allocated, but not initialized) and the state will be updated to READY if initialization success, otherwise the state will return to REGISTERED.

参数
• \texttt{handle} - [in] I2S rx channel handler
• \texttt{pdm\_rx\_cfg} - [in] Configurations for PDM RX mode, including clock, slot and gpio The clock configuration can be generated by the helper macro \texttt{I2S\_PDM\_RX\_CLK\_DEFAULT\_CONFIG} The slot configuration can be generated by the helper macro \texttt{I2S\_PDM\_RX\_SLOT\_DEFAULT\_CONFIG}
### Chapter 2. API 参考

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_OK</td>
<td>Initialize successfully</td>
</tr>
<tr>
<td>ESP_ERR_NO_MEM</td>
<td>No memory for storing the channel information</td>
</tr>
<tr>
<td>ESP_ERR_INVALID_ARG</td>
<td>NULL pointer or invalid configuration</td>
</tr>
<tr>
<td>ESP_ERR_INVALID_STATE</td>
<td>This channel is not registered</td>
</tr>
</tbody>
</table>

**esp_err_t i2s_channel_reconfig_pdm_rx_clock**

```c
i2s_channel_reconfig_pdm_rx_clock(i2s_chan_handle_t handle, const i2s_pdm_rx_clk_config_t *clk_cfg)
```

Reconfigure the I2S clock for PDM RX mode.

**备注:** Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started. This function will not change the state. `i2s_channel_disable` should be called before calling this function if i2s has started.

**参数**
- **handle** [in] I2S rx channel handler
- **clk_cfg** [in] PDM RX mode clock configuration, can be generated by `I2S_PDM_RX_CLK_DEFAULT_CONFIG`

**返回**
- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

**esp_err_t i2s_channel_reconfig_pdm_rx_slot**

```c
i2s_channel_reconfig_pdm_rx_slot(i2s_chan_handle_t handle, const i2s_pdm_rx_slot_config_t *slot_cfg)
```

Reconfigure the I2S slot for PDM RX mode.

**备注:** Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started. This function will not change the state. `i2s_channel_disable` should be called before calling this function if i2s has started.

**参数**
- **handle** [in] I2S rx channel handler
- **slot_cfg** [in] PDM RX mode slot configuration, can be generated by `I2S_PDM_RX_SLOT_DEFAULT_CONFIG`

**返回**
- ESP_OK Set slot successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

**esp_err_t i2s_channel_reconfig_pdm_rx_gpio**

```c
i2s_channel_reconfig_pdm_rx_gpio(i2s_chan_handle_t handle, const i2s_pdm_rx_gpio_config_t *gpio_cfg)
```

Reconfigure the I2S gpio for PDM RX mode.
Chapter 2. API

![Image of a document page]

**Chapter 2. API**

**备注:** Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started. This function will not change the state. ‘i2s_channel_disable’ should be called before calling this function if i2s has started.

**备注:** The input channel handle has to be initialized to PDM RX mode, i.e., ‘i2s_channel_init_pdm_rx_mode’ has been called before reconfiguring.

**参数**
- `handle` - [in] I2S rx channel handler
- `gpio_cfg` - [in] PDM RX mode gpio configuration, specified by user

**返回**
- ESP_OK: Set clock successfully
- ESP_ERR_INVALID_ARG: NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE: This channel is not initialized or not stopped

```c
esp_err_t i2s_channel_init_pdm_rx_mode(i2s_chan_handle_t handle, const i2s_pdm_rx_config_t *pdm_rx_cfg)
```

Initialize i2s channel to PDM RX mode.

**备注:** Only allowed to be called when the channel state is REGISTRED, (i.e., channel has been allocated, but not initialized) and the state will be updated to READY if initialization success, otherwise the state will return to REGISTERED.

**参数**
- `handle` - [in] I2S tx channel handler
- `pdm_tx_cfg` - [in] Configurations for PDM TX mode, including clock, slot and gpio. The clock configuration can be generated by the helper macro `I2S_PDM_TX_CLK_DEFAULT_CONFIG`. The slot configuration can be generated by the helper macro `I2S_PDM_TX_SLOT_DEFAULT_CONFIG`.

**返回**
- ESP_OK: Initialize successfully
- ESP_ERR_NO_MEM: No memory for storing the channel information
- ESP_ERR_INVALID_ARG: NULL pointer or invalid configuration
- ESP_ERR_INVALID_STATE: This channel is not registered

```c
esp_err_t i2s_channel_reconfig_pdm_tx_clock(i2s_chan_handle_t handle, const i2s_pdm_tx_clk_config_t *clk_cfg)
```

Reconfigure the I2S clock for PDM TX mode.

**备注:** Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started. This function will not change the state. ‘i2s_channel_disable’ should be called before calling this function if i2s has started.

**备注:** The input channel handle has to be initialized to PDM TX mode, i.e., ‘i2s_channel_init_pdm_tx_mode’ has been called before reconfiguring.

**参数**
- `handle` - [in] I2S tx channel handler
- `clk_cfg` - [in] PDM TX mode clock configuration, can be generated by `I2S_PDM_TX_CLK_DEFAULT_CONFIG`.
Chapter 2. API

### i2s_channel_reconfig_pdm_tx_slot

```c
esp_err_t i2s_channel_reconfig_pdm_tx_slot(i2s_chan_handle_t handle, const i2s_pdm_tx_slot_config_t *slot_cfg)
```

Reconfigure the I2S slot for PDM TX mode.

**备注:** Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started. This function won’t change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

**参数**

- `handle` - [in] I2S tx channel handler
- `slot_cfg` - [in] PDM TX mode slot configuration, can be generated by `I2S_PDM_TX_SLOT_DEFAULT_CONFIG`

**返回**

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

### i2s_channel_reconfig_pdm_tx_gpio

```c
esp_err_t i2s_channel_reconfig_pdm_tx_gpio(i2s_chan_handle_t handle, const i2s_pdm_tx_gpio_config_t *gpio_cfg)
```

Reconfigure the I2S gpio for PDM TX mode.

**备注:** Only allowed to be called when the channel state is READY, i.e., channel has been initialized, but not started. This function won’t change the state. `i2s_channel_disable` should be called before calling this function if I2S has started.

**参数**

- `handle` - [in] I2S tx channel handler
- `gpio_cfg` - [in] PDM TX mode gpio configuration, specified by user

**返回**

- ESP_OK Set clock successfully
- ESP_ERR_INVALID_ARG NULL pointer, invalid configuration or not PDM mode
- ESP_ERR_INVALID_STATE This channel is not initialized or not stopped

### Structures

```c
struct i2s_pdm_rx_slot_config_t
```

I2S slot configuration for pdm rx mode.
Public Members

\texttt{i2s\_data\_bit\_width\_t data\_bit\_width}
I2S sample data bit width (valid data bits per sample), only support 16 bits for PDM mode

\texttt{i2s\_slot\_bit\_width\_t slot\_bit\_width}
I2S slot bit width (total bits per slot), only support 16 bits for PDM mode

\texttt{i2s\_slot\_mode\_t slot\_mode}
Set mono or stereo mode with I2S\_SLOT\_MODE\_MONO or I2S\_SLOT\_MODE\_STEREO

\texttt{i2s\_pdm\_slot\_mask\_t slot\_mask}
Choose the slots to activate

\texttt{struct i2s\_pdm\_rx\_clk\_config\_t}
I2S clock configuration for pdm rx mode.

Public Members

\texttt{uint32\_t sample\_rate\_hz}
I2S sample rate

\texttt{i2s\_clock\_src\_t clk\_src}
Choose clock source

\texttt{i2s\_mclk\_multiple\_t mclk\_multiple}
The multiple of mclk to the sample rate

\texttt{i2s\_pdm\_dsr\_t dn\_sample\_mode}
Down-sampling rate mode

\texttt{struct i2s\_pdm\_rx\_gpio\_config\_t}
I2S PDM tx mode GPIO pins configuration.

Public Members

\texttt{gpio\_num\_t clk}
PDM clk pin, output

\texttt{gpio\_num\_t din}
DATA pin 0, input

\texttt{gpio\_num\_t dins[n(U)]}
DATA pins, input, only take effect when corresponding I2S\_PDM\_RX\_LINE\_SLOT\_xxx is enabled in i2s\_pdm\_rx\_slot\_config\_t::slot\_mask

\texttt{uint32\_t clk\_inv}
Set 1 to invert the clk output
struct `i2s_pdm_rx_gpio_config_t`::[anonymous] `invert_flags`
  GPIO pin invert flags

struct `i2s_pdm_rx_config_t`
  I2S PDM RX mode major configuration that including clock/slot/gpio configuration.

**Public Members**

`i2s_pdm_rx_clk_config_t` `clk_cfg`
  PDM RX clock configurations, can be generated by macro `I2S_PDM_RX_CLK_DEFAULT_CONFIG`

`i2s_pdm_rx_slot_config_t` `slot_cfg`
  PDM RX slot configurations, can be generated by macro `I2S_PDM_RX_SLOT_DEFAULT_CONFIG`

`i2s_pdm_rx_gpio_config_t` `gpio_cfg`
  PDM RX slot configurations, specified by user

struct `i2s_pdm_tx_slot_config_t`
  I2S slot configuration for pdm tx mode.

**Public Members**

`i2s_data_bit_width_t` `data_bit_width`
  I2S sample data bit width (valid data bits per sample), only support 16 bits for PDM mode

`i2s_slot_bit_width_t` `slot_bit_width`
  I2S slot bit width (total bits per slot), only support 16 bits for PDM mode

`i2s_slot_mode_t` `slot_mode`
  Set mono or stereo mode with I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO For PDM TX mode, mono means the data buffer only contains one slot data, Stereo means the data buffer contains two slots data

`i2s_pdm_slot_mask_t` `slot_mask`
  Slot mask to choose left or right slot

`uint32_t` `sd_prescale`
  Sigma-delta filter prescale

`i2s_pdm_sig_scale_t` `sd_scale`
  Sigma-delta filter scaling value

`i2s_pdm_sig_scale_t` `hp_scale`
  High pass filter scaling value

`i2s_pdm_sig_scale_t` `lp_scale`
  Low pass filter scaling value
Chapter 2. API 参考

\textit{i2s_pdm_sig_scale_t} \textbf{sinc\_scale}
Sinc filter scaling value

\textbf{struct i2s_pdm_tx_clk_config_t}
I2S clock configuration for pdm tx mode.

\textbf{Public Members}

\textbf{uint32_t} \textit{sample\_rate\_hz}
I2S sample rate

\textbf{i2s_clock_src_t} \textit{clk\_src}
Choose clock source

\textbf{i2s_mclk_multiple_t} \textit{mclk\_multiple}
The multiple of mclk to the sample rate

\textbf{uint32_t} \textit{up\_sample\_fp}
Up-sampling param fp

\textbf{uint32_t} \textit{up\_sample\_fs}
Up-sampling param fs

\textbf{struct i2s_pdm_tx_gpio_config_t}
I2S PDM tx mode GPIO pins configuration.

\textbf{Public Members}

\textbf{gpio\_num\_t} \textit{clk}
PDM clk pin, output

\textbf{gpio\_num\_t} \textit{dout}
DATA pin, output

\textbf{uint32\_t} \textit{clk\_inv}
Set 1 to invert the clk output

\textbf{struct i2s_pdm_tx_gpio_config_t::[anonymous]} \textbf{invert\_flags}
GPIO pin invert flags

\textbf{struct i2s_pdm_tx_config_t}
I2S PDM TX mode major configuration that including clock/slot/gpio configuration.

\textbf{Public Members}
Chapter 2. API 参考

**i2s_pdm_tx_clk_config_t clk_cfg**

PDM TX clock configurations, can be generated by macro `I2S_PDM_TX_CLK_DEFAULT_CONFIG`

**i2s_pdm_tx_slot_config_t slot_cfg**

PDM TX slot configurations, can be generated by macro `I2S_PDM_TX_SLOT_DEFAULT_CONFIG`

**i2s_pdm_tx_gpio_config_t gpio_cfg**

PDM TX gpio configurations, specified by user

### Macros

**I2S_PDM_RX_SLOT_DEFAULT_CONFIG** (bits_per_sample, mono_or_stereo)

PDM format in 2 slots (RX)

This file is specified for I2S PDM communication mode Features:

- Only support PDM tx/rx mode
- Fixed to 2 slots
- Data bit width only support 16 bits

参数

- **bits_per_sample** - i2s data bit width, only support 16 bits for PDM mode
- **mono_or_stereo** - I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

**I2S_PDM_RX_CLK_DEFAULT_CONFIG** (rate)

i2s default pdm rx clock configuration

参数

- **rate** - sample rate

**I2S_PDM_TX_SLOT_DEFAULT_CONFIG** (bits_per_sample, mono_or_stereo)

PDM style in 2 slots (TX)

参数

- **bits_per_sample** - i2s data bit width, only support 16 bits for PDM mode
- **mono_or_stereo** - I2S_SLOT_MODE_MONO or I2S_SLOT_MODE_STEREO

**I2S_PDM_TX_CLK_DEFAULT_CONFIG** (rate)

i2s default pdm tx clock configuration

### 备注: TX PDM can only be set to the following two up-sampling rate configurations:

1. \( fp = 960, fs = \text{sample\_rate}\_hz\_/100, \) in this case, \( Fpdm = 128*48000 \)
2. \( fp = 960, fs = 480, \) in this case, \( Fpdm = 128*Fpcm = 128*\text{sample\_rate}\_hz \)

If the pdm receiver do not care the pdm serial clock, it’s recommended set \( Fpdm = 128*48000 \). Otherwise, the second configuration should be adopted.

参数

- **rate** - sample rate

### I2S Driver

**Header File**

- components/driver/include/driver/i2s_common.h
Chapter 2. API 参考

Functions

```c
esp_err_t i2s_new_channel(const i2s_chan_config_t *chan_cfg, i2s_chan_handle_t *ret_tx_handle, i2s_chan_handle_t *ret_rx_handle)
```

Allocate new I2S channel(s)

**备注**: The new created I2S channel handle will be REGISTERED state after it is allocated successfully.

**备注**: When the port id in channel configuration is I2S_NUM_AUTO, driver will allocate I2S port automatically on one of the i2s controller, otherwise driver will try to allocate the new channel on the selected port.

**备注**: If both tx_handle and rx_handle are not NULL, it means this I2S controller will work at full-duplex mode, the rx and tx channels will be allocated on a same I2S port in this case. Note that some configurations of tx/rx channel are shared on ESP32 and ESP32S2, so please make sure they are working at same condition and under same status(start/stop). Currently, full-duplex mode can’t guarantee tx/rx channels write/read synchronously, they can only share the clock signals for now.

**备注**: If tx_handle OR rx_handle is NULL, it means this I2S controller will work at simplex mode. For ESP32 and ESP32S2, the whole I2S controller (i.e. both rx and tx channel) will be occupied, even if only one of rx or tx channel is registered. For the other targets, another channel on this controller will still available.

### Parameters
- `chan_cfg` [in] I2S controller channel configurations
- `ret_tx_handle` [out] I2S channel handler used for managing the sending channel(optional)
- `ret_rx_handle` [out] I2S channel handler used for managing the receiving channel(optional)

### Return
- ESP_OK Allocate new channel(s) success
- ESP_ERR_NOT_SUPPORTED The communication mode is not supported on the current chip
- ESP_ERR_INVALID_ARG NULL pointer or illegal parameter in i2s_chan_config_t
- ESP_ERR_NOT_FOUND No available I2S channel found

```c
esp_err_t i2s_del_channel(i2s_chan_handle_t handle)
```

Delete the i2s channel.

**备注**: Only allowed to be called when the i2s channel is at REGISTERED or READY state (i.e., it should stop before deleting it).

**备注**: Resource will be free automatically if all channels in one port are deleted.

### Parameters
- `handle` [in] I2S channel handler
  - ESP_OK Delete successfully
  - ESP_ERR_INVALID_ARG NULL pointer

```c
esp_err_t i2s_channel_get_info(i2s_chan_handle_t handle, i2s_chan_info_t *chan_info)
```

Get I2S channel information.
### Chapter 2. API 参考

#### 参数
- handle - [in] I2S channel handler
- chan_info - [out] I2S channel basic information

#### 返回
- ESP_OK Get i2s channel information success
- ESP_ERR_NOT_FOUND The input handle doesn’t match any registered I2S channels, it may not an i2s channel handle or not available any more
- ESP_ERR_INVALID_ARG The input handle or chan_info pointer is NULL

**esp_err_t i2s_channel_enable (i2s_chan_handle_t handle)**

Enable the i2s channel.

**备注**: Only allowed to be called when the channel state is READY, (i.e., channel has been initialized, but not started) the channel will enter RUNNING state once it is enabled successfully.

**备注**: Enable the channel can start the I2S communication on hardware. It will start outputting blck and ws signal. For mclk signal, it will start to output when initialization is finished

#### 参数
- handle - [in] I2S channel handler
- ESP_OK Start successfully
- ESP_ERR_INVALID_ARG NULL pointer
- ESP_ERR_INVALID_STATE This channel has not initialized or already started

**esp_err_t i2s_channel_disable (i2s_chan_handle_t handle)**

Disable the i2s channel.

**备注**: Only allowed to be called when the channel state is READY / RUNNING, (i.e., channel has been initialized) the channel will enter READY state once it is disabled successfully.

**备注**: Disable the channel can stop the I2S communication on hardware. It will stop blck and ws signal but not mclk signal

#### 参数
- handle - [in] I2S channel handler
- ESP_OK Stop successfully
- ESP_ERR_INVALID_ARG NULL pointer
- ESP_ERR_INVALID_STATE This channel has not stated

**esp_err_t i2s_channel_write (i2s_chan_handle_t handle, const void *src, size_t size, size_t *bytes_written, uint32_t timeout_ms)**

I2S write data.

**备注**: Only allowed to be called when the channel state is RUNNING, (i.e., tx channel has been started and is not writing now) but the RUNNING only stands for the software state, it doesn’t mean there is no the signal transporting on line.

#### 参数
- handle - [in] I2S channel handler
- src - [in] The pointer of sent data buffer
- size - [in] Max data buffer length
Chapter 2. API Reference

- **bytes_written** - [out] Byte number that actually be sent
- **timeout_ms** - [in] Max block time

**Returned**:
- ESP_OK Write successfully
- ESP_ERR_INVALID_ARG NULL pointer or this handle is not tx handle
- ESP_ERR_TIMEOUT Writing timeout, no writing event received from ISR within ticks_to_wait
- ESP_ERR_INVALID_STATE I2S is not ready to write

```c
esp_err_t i2s_channel_read(i2s_chan_handle_t handle, void *dest, size_t size, size_t *bytes_read, uint32_t timeout_ms)
```

I2S read data.

**备注**: Only allowed to be called when the channel state is RUNNING but the RUNNING only stands for the software state, it doesn’t mean there is no the signal transporting on line.

**Parameters**:
- **handle** - [in] I2S channel handler
- **dest** - [in] The pointer of receiving data buffer
- **size** - [in] Max data buffer length
- **bytes_read** - [out] Byte number that actually be read
- **timeout_ms** - [in] Max block time

**Returned**:
- ESP_OK Read successfully
- ESP_ERR_INVALID_ARG NULL pointer or this handle is not rx handle
- ESP_ERR_TIMEOUT Reading timeout, no reading event received from ISR within ticks_to_wait
- ESP_ERR_INVALID_STATE I2S is not ready to read

```c
esp_err_t i2s_channel_register_event_callback(i2s_chan_handle_t handle, const i2s_event_callbacks_t *callbacks, void *user_data)
```

Set event callbacks for I2S channel.

**备注**: Only allowed to be called when the channel state is REGISTARED / READY, (i.e., before channel starts)

**备注**: User can deregister a previously registered callback by calling this function and setting the callback member in the callbacks structure to NULL.

**备注**: When CONFIG_I2S_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The user_data should also reside in SRAM or internal RAM as well.

**Parameters**:
- **handle** - [in] I2S channel handler
- **callbacks** - [in] Group of callback functions
- **user_data** - [in] User data, which will be passed to callback functions directly

**Returned**:
- ESP_OK Set event callbacks successfully
- ESP_ERR_INVALID_ARG Set event callbacks failed because of invalid argument
ESP_ERR_INVALID_STATE – Set event callbacks failed because the current channel state is not REGISTERED or READY

**Structures**

```c
define i2s_event_callbacks_t
    Group of I2S callbacks.
```

备注: The callbacks are all running under ISR environment

备注: When CONFIG_I2S_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

**Public Members**

```c
typedef i2s_isr_callback_t on_recv
    Callback of data received event, only for rx channel The event data includes DMA buffer address and size that just finished receiving data
```

```c
typedef i2s_isr_callback_t on_recv_q_ovf
    Callback of receiving queue overflowed event, only for rx channel The event data includes buffer size that has been overwritten
```

```c
typedef i2s_isr_callback_t on_sent
    Callback of data sent event, only for tx channel The event data includes DMA buffer address and size that just finished sending data
```

```c
typedef i2s_isr_callback_t on_send_q_ovf
    Callback of sending queue overflowed event, only for tx channel The event data includes buffer size that has been overwritten
```

```c
struct i2s_chan_config_t
    I2S controller channel configuration.
```

**Public Members**

```c
typedef i2s_port_t id
    I2S port id
```

```c
typedef i2s_role_t role
    I2S role, I2S_ROLE_MASTER or I2S_ROLE_SLAVE
```

```c
uint32_t dma_desc_num
    I2S DMA buffer number, it is also the number of DMA descriptor
```

```c
uint32_t dma_frame_num
    I2S frame number in one DMA buffer. One frame means one-time sample data in all slots, it should be the multiple of ‘3’ when the data bit width is 24.
```
bool auto_clear

Set to auto clear DMA TX buffer, i2s will always send zero automatically if no data to send

struct i2s_chan_info_t
I2S channel information.

Public Members

i2s_port_t id
I2S port id

i2s_role_t role
I2S role, I2S_ROLE_MASTER or I2S_ROLE_SLAVE

i2s_dir_t dir
I2S channel direction

i2s_comm_mode_t mode
I2S channel communication mode

i2s_chan_handle_t pair_chan
I2S pair channel handle in duplex mode, always NULL in simplex mode

Macros

I2S_CHANNEL_DEFAULT_CONFIG (i2s_num, i2s_role)
get default I2S property

I2S_GPIO_UNUSED
Used in i2s_gpio_config_t for signals which are not used

I2S Types

Header File

* components/driver/include/driver/i2s_types.h

Structures

struct i2s_event_data_t
Event structure used in I2S event queue.

Public Members

void *data
The pointer of DMA buffer that just finished sending or receiving for on_recv and on_sent callback
NULL for on_recv_q_ovf and on_send_q_ovf callback
size_t size

The buffer size of DMA buffer when success to send or receive, also the buffer size that dropped when queue overflow. It is related to the dma_frame_num and data_bit_width, typically it is fixed when data_bit_width is not changed.

Type Definitions

typedef struct i2s_channel_obj_t *i2s_chan_handle_t
i2s channel object handle, the control unit of the i2s driver

typedef bool (*i2s_isr_callback_t)(i2s_chan_handle_t handle, i2s_event_data_t *event, void *user_ctx)
I2S event callback.

Param handle [in] I2S channel handle, created from i2s_new_channel()
Param event [in] I2S event data
Param user_ctx [in] User registered context, passed from i2s_channel_register_event_callback()
Return Whether a high priority task has been waken up by this callback function

Enumerations

enum i2s_port_t
I2S controller port number, the max port number is (SOC_I2S_NUM -1).

Values:

enumerator I2S_NUM_0
I2S controller port 0

enumerator I2S_NUM_1
I2S controller port 1

enumerator I2S_NUM_AUTO
Select whichever port is available

enum i2s_comm_mode_t
I2S controller communication mode.

Values:

enumerator I2S_COMM_MODE_STD
I2S controller using standard communication mode, support philips/MSB/PCM format

enumerator I2S_COMM_MODE_PDM
I2S controller using PDM communication mode, support PDM output or input

enumerator I2S_COMM_MODE_NONE
Unspecified I2S controller mode

enum i2s_mclk_multiple_t
The multiple of mclk to sample rate.

Values:
enumerator **I2S_MCLK_MULTIPLE_128**
\[
mclk = \text{sample} \cdot \text{rate} \times 128
\]

enumerator **I2S_MCLK_MULTIPLE_256**
\[
mclk = \text{sample} \cdot \text{rate} \times 256
\]

enumerator **I2S_MCLK_MULTIPLE_384**
\[
mclk = \text{sample} \cdot \text{rate} \times 384
\]

enumerator **I2S_MCLK_MULTIPLE_512**
\[
mclk = \text{sample} \cdot \text{rate} \times 512
\]

**Header File**

- components/hal/include/hal/i2s_types.h

**Type Definitions**

typedef `soc_periph_i2s_clk_src_t` **i2s_clock_src_t**

I2S clock source

**Enumerations**

enum **i2s_slot_mode_t**

I2S channel slot mode.

*Values:*

enumerator **I2S_SLOT_MODE_MONO**

I2S channel slot format mono, transmit same data in all slots for tx mode, only receive the data in the first slots for rx mode.

enumerator **I2S_SLOT_MODE_STEREO**

I2S channel slot format stereo, transmit different data in different slots for tx mode, receive the data in all slots for rx mode.

enum **i2s_dir_t**

I2S channel direction.

*Values:*

enumerator **I2S_DIR_RX**

I2S channel direction RX

enumerator **I2S_DIR_TX**

I2S channel direction TX

enum **i2s_role_t**

I2S controller role.

*Values:*
enumerator **I2S_ROLE_MASTER**

I2S controller master role, bclk and ws signal will be set to output

enumerator **I2S_ROLE_SLAVE**

I2S controller slave role, bclk and ws signal will be set to input

**enum i2s_data_bit_width_t**

Available data bit width in one slot.

*Values:*

enumerator **I2S_DATA_BIT_WIDTH_8BIT**

I2S channel data bit-width: 8

enumerator **I2S_DATA_BIT_WIDTH_16BIT**

I2S channel data bit-width: 16

enumerator **I2S_DATA_BIT_WIDTH_24BIT**

I2S channel data bit-width: 24

enumerator **I2S_DATA_BIT_WIDTH_32BIT**

I2S channel data bit-width: 32

**enum i2s_slot_bit_width_t**

Total slot bit width in one slot.

*Values:*

enumerator **I2S_SLOT_BIT_WIDTH_AUTO**

I2S channel slot bit-width equals to data bit-width

enumerator **I2S_SLOT_BIT_WIDTH_8BIT**

I2S channel slot bit-width: 8

enumerator **I2S_SLOT_BIT_WIDTH_16BIT**

I2S channel slot bit-width: 16

enumerator **I2S_SLOT_BIT_WIDTH_24BIT**

I2S channel slot bit-width: 24

enumerator **I2S_SLOT_BIT_WIDTH_32BIT**

I2S channel slot bit-width: 32

**enum i2s_pdm_dsr_t**

I2S PDM RX down-sampling mode.

*Values:*

enumerator **I2S_PDM_DSR_8S**

downsampling number is 8 for PDM RX mode
enumerator I2S_PDM_DSR_16S
downsampling number is 16 for PDM RX mode

enumerator I2S_PDM_DSR_MAX

enum i2s_pdm_sig_scale_t
pdm tx signal scaling mode
Values:

enumerator I2S_PDM_SIG_SCALING_DIV_2
I2S TX PDM signal scaling: /2

enumerator I2S_PDM_SIG_SCALING_MUL_1
I2S TX PDM signal scaling: x1

enumerator I2S_PDM_SIG_SCALING_MUL_2
I2S TX PDM signal scaling: x2

enumerator I2S_PDM_SIG_SCALING_MUL_4
I2S TX PDM signal scaling: x4

enum i2s_std_slot_mask_t
I2S slot select in standard mode.
Values:

enumerator I2S_STD SLOT_LEFT
I2S transmits or receives left slot

enumerator I2S_STD SLOT_RIGHT
I2S transmits or receives right slot

enumerator I2S_STD SLOT BOTH
I2S transmits or receives both left and right slot

enum i2s_pdm_slot_mask_t
I2S slot select in PDM mode.
Values:

enumerator I2S_PDM SLOT RIGHT
I2S PDM only transmits or receives the PDM device whose ‘select’ pin is pulled up

enumerator I2S_PDM SLOT LEFT
I2S PDM only transmits or receives the PDM device whose ‘select’ pin is pulled down
enumerator I2S_PDM_SLOT_BOTH

I2S PDM transmits or receives both two slots

2.6.10 LCD

Introduction

ESP chips can generate various kinds of timings that needed by common LCDs on the market, like SPI LCD, 80 LCD (a.k.a Intel 8080 parallel LCD), RGB/RGB LCD, I2C LCD, etc. The esp_lcd component is officially to support those LCDs with a group of universal APIs across chips.

Functional Overview

In esp_lcd, an LCD panel is represented by esp_lcd_panel_handle_t, which plays the role of an abstract frame buffer, regardless of the frame memory is allocated inside ESP chip or in external LCD controller. Based on the location of the frame buffer, the LCD panel allocation functions are mainly grouped into the following categories:

- **RGB LCD panel** - is simply based on a group of specific synchronous signals indicating where to start and stop a frame.
- **Controller based LCD panel** involves multiple steps to get a panel handle, like bus allocation, IO device registration and controller driver install.

After we get the LCD handle, the remaining LCD operations are similar for different LCD interfaces and vendors.

Application Example

LCD examples are located under: peripherals/lcd:

- Universal SPI LCD example with SPI touch - peripherals/lcd/spi_lcd_touch
- Jpeg decoding and LCD display - peripherals/lcd/tjgpd
- i80 controller based LCD and LVGL animation UI - peripherals/lcd/i80_controller
- I2C interfaced OLED display scrolling text - peripherals/lcd/i2c_oled

Other LCD drivers

Drivers for some LCD and touch controllers are available in IDF Component Registry. The list of available and planned drivers with links is in this table.

API Reference

Header File

- components/hal/include/hal/lcd_types.h

Type Definitions

typedef soc_periph_lcd_clk_src_t lcd_clock_source_t

LCD clock source.
Enumerations

enum *lcd_color_rgb_endian_t*

  RGB color endian.

  *Values:*

  enumerator *LCD_RGB_ENDIAN_RGB*
  
  RGB data endian: RGB

  enumerator *LCD_RGB_ENDIAN_BGR*
  
  RGB data endian: BGR

enum *lcd_color_space_t*

  LCD color space.

  *Values:*

  enumerator *LCD_COLOR_SPACE_RGB*
  
  Color space: RGB

  enumerator *LCD_COLOR_SPACE_YUV*
  
  Color space: YUV

enum *lcd_color_range_t*

  LCD color range.

  *Values:*

  enumerator *LCD_COLOR_RANGE_LIMIT*
  
  Limited color range

  enumerator *LCD_COLOR_RANGE_FULL*
  
  Full color range

enum *lcd_yuv_sample_t*

  YUV sampling method.

  *Values:*

  enumerator *LCD_YUV_SAMPLE_422*
  
  YUV 4:2:2 sampling

  enumerator *LCD_YUV_SAMPLE_420*
  
  YUV 4:2:0 sampling

  enumerator *LCD_YUV_SAMPLE_411*
  
  YUV 4:1:1 sampling

enum *lcd_yuv_conv_std_t*

  The standard used for conversion between RGB and YUV.

  *Values:*

  enumerator *LCD_YUV_CONV_STD_1087*
  
  YUV 1087 conversion standard
Chapter 2. API

enumerator **LCD_YUV_CONV_STD_BT601**
YUV<->RGB conversion standard: BT.601

enumerator **LCD_YUV_CONV_STD_BT709**
YUV<->RGB conversion standard: BT.709

**Header File**

- components/esp_lcd/include/esp_lcd_types.h

**Type Definitions**

typedef struct esp_lcd_panel_io_t * **esp_lcd_panel_io_handle_t**
Type of LCD panel IO handle

typedef struct esp_lcd_panel_t * **esp_lcd_panel_handle_t**
Type of LCD panel handle

**Header File**

- components/esp_lcd/include/esp_lcd_panel_io.h

**Functions**

```c
esp_err_t esp_lcd_panel_io_rx_param(esp_lcd_panel_io_handle_t io, int lcd_cmd, void *param, size_t param_size)
```
Transmit LCD command and receive corresponding parameters.

**备注:** Commands sent by this function are short, so they are sent using polling transactions. The function does not return before the command transfer is completed. If any queued transactions sent by `esp_lcd_panel_io_tx_color()` are still pending when this function is called, this function will wait until they are finished and the queue is empty before sending the command(s).

```c
io – [in] LCD panel IO handle, which is created by other factory API like `esp_lcd_new_panel_io_spi()`
```

```c
lcd_cmd – [in] The specific LCD command, set to -1 if no command needed
```

```c
param – [out] Buffer for the command data
```

```c
param_size – [in] Size of param buffer
```

**返回**

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_SUPPORTED if read is not supported by transport
- ESP_OK on success

```c
esp_err_t esp_lcd_panel_io_tx_param(esp_lcd_panel_io_handle_t io, int lcd_cmd, const void *param, size_t param_size)
```
Transmit LCD command and corresponding parameters.

**备注:** Commands sent by this function are short, so they are sent using polling transactions. The function does not return before the command transfer is completed. If any queued transactions sent by `esp_lcd_panel_io_tx_color()` are still pending when this function is called, this function will wait until they are finished and the queue is empty before sending the command(s).
### Chapter 2. API 参考

**参数**

- **io** – [in] LCD panel IO handle, which is created by other factory API like esp_lcd_new_panel_io_spi()
- **lcd_cmd** – [in] The specific LCD command (set to -1 if no command needed - only in SPI and I2C)
- **param** – [in] Buffer that holds the command specific parameters, set to NULL if no parameter is needed for the command
- **param_size** – [in] Size of param in memory, in bytes, set to zero if no parameter is needed for the command

**返回**

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

```c
esp_err_t esp_lcd_panel_io_tx_color(esp_lcd_panel_io_handle_t io, int lcd_cmd, const void* color, size_t color_size)
```

Transmit LCD RGB data.

**备注:** This function will package the command and RGB data into a transaction, and push into a queue. The real transmission is performed in the background (DMA+interrupt). The caller should take care of the lifecycle of the color buffer. Recycling of color buffer should be done in the callback on_color_trans_done().

### 参数

- **io** – [in] LCD panel IO handle, which is created by factory API like esp_lcd_new_panel_io_spi()
- **lcd_cmd** – [in] The specific LCD command, set to -1 if no command needed
- **color** – [in] Buffer that holds the RGB color data
- **color_size** – [in] Size of color in memory, in bytes

**返回**

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

```c
esp_err_t esp_lcd_panel_io_del(esp_lcd_panel_io_handle_t io)
```

Destroy LCD panel IO handle (deinitialize panel and free all corresponding resource)

**参数**

- **io** – [in] LCD panel IO handle, which is created by factory API like esp_lcd_new_panel_io_spi()

**返回**

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

```c
esp_err_t esp_lcd_panel_io_register_event_callbacks(esp_lcd_panel_io_handle_t io, const esp_lcd_panel_io_callbacks_t* cbs, void* user_ctx)
```

Register LCD panel IO callbacks.

**参数**

- **io** – [in] LCD panel IO handle, which is created by factory API like esp_lcd_new_panel_io_spi()
- **cbs** – [in] structure with all LCD panel IO callbacks
- **user_ctx** – [in] User private data, passed directly to callback’s user_ctx

**返回**

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

```c
esp_err_t esp_lcd_new_panel_io_spi(esp_spi_bus_handle_t bus, const esp_lcd_panel_io_spi_config_t* io_config, esp_lcd_panel_io_handle_t* ret_io)
```

Create LCD panel IO handle, for SPI interface.
### Chapter 2. API 参考

**参数**
- `bus` - [in] SPI bus handle
- `io_config` - [in] IO configuration, for SPI interface
- `ret_io` - [out] Returned IO handle

**返回**
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

```c
esp_err_t esp_lcd_new_panel_io_i2c(esp_lcd_i2c_bus_handle_t bus, const
                                 esp_lcd_panel_io_i2c_config_t *io_config,
                                 esp_lcd_panel_io_handle_t *ret_io)
```

Create LCD panel IO handle, for I2C interface.

**参数**
- `bus` - [in] I2C bus handle
- `io_config` - [in] IO configuration, for I2C interface
- `ret_io` - [out] Returned IO handle

**返回**
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

```c
esp_err_t esp_lcd_new_i80_bus(const esp_lcd_i80_bus_config_t *bus_config,
                              esp_lcd_i80_bus_handle_t *ret_bus)
```

Create Intel 8080 bus handle.

**参数**
- `bus_config` - [in] Bus configuration
- `ret_bus` - [out] Returned bus handle

**返回**
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_NOT_FOUND if no free bus is available
- ESP_OK on success

```c
esp_err_t esp_lcd_del_i80_bus(esp_lcd_i80_bus_handle_t bus)
```

Destroy Intel 8080 bus handle.

**参数**
- `bus` - [in] Intel 8080 bus handle, created by esp_lcd_new_i80_bus()

**返回**
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if there still be some device attached to the bus
- ESP_OK on success

```c
esp_err_t esp_lcd_new_panel_io_i80(esp_lcd_i80_bus_handle_t bus, const
                                   esp_lcd_panel_io_i80_config_t *io_config,
                                   esp_lcd_panel_io_handle_t *ret_io)
```

Create LCD panel IO, for Intel 8080 interface.

**参数**
- `bus` - [in] Intel 8080 bus handle, created by esp_lcd_new_i80_bus()
- `io_config` - [in] IO configuration, for i80 interface
- `ret_io` - [out] Returned panel IO handle

**返回**
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_SUPPORTED if some configuration can’t be satisfied, e.g. pixel clock out of the range
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success
Chapter 2. API Reference

Structures

struct esp_lcd_panel_io_event_data_t
Type of LCD panel IO event data.

struct esp_lcd_panel_io_callbacks_t
Type of LCD panel IO callbacks.

Public Members

esp_lcd_panel_io_color_trans_done_cb_t on_color_trans_done
Callback invoked when color data transfer has finished

struct esp_lcd_panel_io_spi_config_t
Panel IO configuration structure, for SPI interface.

Public Members

int cs_gpio_num
GPIO used for CS line

int dc_gpio_num
GPIO used to select the D/C line, set this to -1 if the D/C line is not used

int spi_mode
Traditional SPI mode (0~3)

unsigned int pclk_hz
Frequency of pixel clock

size_t trans_queue_depth
Size of internal transaction queue

esp_lcd_panel_io_color_trans_done_cb_t on_color_trans_done
Callback invoked when color data transfer has finished

void *user_ctx
User private data, passed directly to on_color_trans_done’s user_ctx

int lcd_cmd_bits
Bit-width of LCD command

int lcd_param_bits
Bit-width of LCD parameter

unsigned int dc_low_on_data
If this flag is enabled, DC line = 0 means transfer data, DC line = 1 means transfer command; vice versa
unsigned int `octal_mode`
transmit with octal mode (8 data lines), this mode is used to simulate Intel 8080 timing

unsigned int `sio_mode`
Read and write through a single data line (MOSI)

unsigned int `lsb_first`
transmit LSB bit first

unsigned int `cs_high_active`
CS line is high active

struct `esp_lcd_panel_io_spi_config_t`: [anonymous] `flags`
Extra flags to fine-tune the SPI device

struct `esp_lcd_panel_io_i2c_config_t`
Panel IO configuration structure, for I2C interface.

**Public Members**

```
uint32_t `dev_addr`
I2C device address
```

```
estp_lcd_panel_io_color_trans_doneCb_t `on_color_trans_done`
Callback invoked when color data transfer has finished
```

```
void *`user_ctx`
User private data, passed directly to on_color_trans_done’s user_ctx
```

```
size_t `control_phase_bytes`
I2C LCD panel will encode control information (e.g. D/C selection) into control phase, in several bytes
```

```
unsigned int `dc_bit_offset`
Offset of the D/C selection bit in control phase
```

```
int `lcd_cmd_bits`
Bit-width of LCD command
```

```
int `lcd_param_bits`
Bit-width of LCD parameter
```

```
unsigned int `dc_low_on_data`
If this flag is enabled, DC line = 0 means transfer data, DC line = 1 means transfer command; vice versa
```

```
unsigned int `disable_control_phase`
If this flag is enabled, the control phase isn’t used
```
struct esp_lcd_panel_io_i2c_config_t::[anonymous] flags
    Extra flags to fine-tune the I2C device

struct esp_lcd_i80_bus_config_t
    LCD Intel 8080 bus configuration structure.

Public Members

int dc_gpio_num
    GPIO used for D/C line

int wr_gpio_num
    GPIO used for WR line

lcd_clock_source_t clk_src
    Clock source for the I80 LCD peripheral

int data_gpio_nums[24]
    GPIOs used for data lines

size_t bus_width
    Number of data lines, 8 or 16

size_t max_transfer_bytes
    Maximum transfer size, this determines the length of internal DMA link

size_t psram_trans_align
    DMA transfer alignment for data allocated from PSRAM

size_t sram_trans_align
    DMA transfer alignment for data allocated from SRAM

struct esp_lcd_panel_io_i80_config_t
    Panel IO configuration structure, for intel 8080 interface.

Public Members

int cs_gpio_num
    GPIO used for CS line, set to -1 will declaim exclusively use of I80 bus

uint32_t pclk_hz
    Frequency of pixel clock

size_t trans_queue_depth
    Transaction queue size, larger queue, higher throughput
### Chapter 2. API 参考

**esp_lcd_panel_io_color_trans_done cb_t on_color_trans_done**

Callback invoked when color data was transferred done

```c
void *user_ctx
```

User private data, passed directly to on_color_trans_done’s user_ctx

```c
int lcd_cmd_bits
```

Bit-width of LCD command

```c
int lcd_param_bits
```

Bit-width of LCD parameter

```c
unsigned int dc_idle_level
```

Level of DC line in IDLE phase

```c
unsigned int dc_cmd_level
```

Level of DC line in CMD phase

```c
unsigned int dc_dummy_level
```

Level of DC line in DUMMY phase

```c
unsigned int dc_data_level
```

Level of DC line in DATA phase

```c
struct esp_lcd_panel_io_i80_config_t::[anonymous] dc_levels
```

Each i80 device might have its own D/C control logic

```c
unsigned int cs_active_high
```

If set, a high level of CS line will select the device, otherwise, CS line is low level active

```c
unsigned int reverse_color_bits
```

Reverse the data bits, D[N:0] -> D[0:N]

```c
unsigned int swap_color_bytes
```

Swap adjacent two color bytes

```c
unsigned int pclk_active_neg
```

The display will write data lines when there’s a falling edge on WR signal (a.k.a the PCLK)

```c
unsigned int pclk_idle_low
```

The WR signal (a.k.a the PCLK) stays at low level in IDLE phase

```c
struct esp_lcd_panel_io_i80_config_t::[anonymous] flags
```

Panel IO config flags

### Type Definitions

```c
typedef void *esp_lcd_spi_bus_handle_t
```

Type of LCD SPI bus handle
typedef void *esp_lcd_i2c_bus_handle_t

Type of LCD I2C bus handle

typedef struct esp_lcd_i80_bus_t *esp_lcd_i80_bus_handle_t

Type of LCD intel 8080 bus handle

typedef bool (*esp_lcd_panel_io_color_trans_done_cb_t)(esp_lcd_panel_io_handle_t panel_io, esp_lcd_panel_io_event_data_t *edata, void *user_ctx)

Declare the prototype of the function that will be invoked when panel IO finishes transferring color data.

  Param panel_io [in] LCD panel IO handle, which is created by factory API like esp_lcd_new_panel_io_spi()
  Param edata [in] Panel IO event data, fed by driver
  Param user_ctx [in] User data, passed from esp_lcd_panel_io_xxx_config_t
  Return Whether a high priority task has been waken up by this function

Header File

• components/esp_lcd/include/esp_lcd_panel_ops.h

Functions

esp_err_t esp_lcd_panel_reset (esp_lcd_panel_handle_t panel)

Reset LCD panel.

备注: Panel reset must be called before attempting to initialize the panel using esp_lcd_panel_init().

参数 panel -[in] LCD panel handle, which is created by other factory API like esp_lcd_new_panel_st7789()

返回
  • ESP_OK on success

esp_err_t esp_lcd_panel_init (esp_lcd_panel_handle_t panel)

Initialize LCD panel.

备注: Before calling this function, make sure the LCD panel has finished the reset stage by esp_lcd_panel_reset().

参数 panel -[in] LCD panel handle, which is created by other factory API like esp_lcd_new_panel_st7789()

返回
  • ESP_OK on success

esp_err_t esp_lcd_panel_del (esp_lcd_panel_handle_t panel)

Deinitialize the LCD panel.

参数 panel -[in] LCD panel handle, which is created by other factory API like esp_lcd_new_panel_st7789()

返回
  • ESP_OK on success

esp_err_t esp_lcd_panel_draw_bitmap (esp_lcd_panel_handle_t panel, int x_start, int y_start, int x_end, int y_end, const void *color_data)

Draw bitmap on LCD panel.
### Chapter 2. API 参考

#### 参数

- **panel** - [in] LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **x_start** - [in] Start index on x-axis (x_start included)
- **y_start** - [in] Start index on y-axis (y_start included)
- **x_end** - [in] End index on x-axis (x_end not included)
- **y_end** - [in] End index on y-axis (y_end not included)
- **color_data** - [in] RGB color data that will be dumped to the specific window range

#### 返回

- ESP_OK on success

```c
esp_err_t esp_lcd_panel_mirror(esp_lcd_panel_handle_t panel, bool mirror_x, bool mirror_y)
```

Mirror the LCD panel on specific axis.

**备注：** Combined with `esp_lcd_panel_swap_xy()`, one can realize screen rotation.

#### 参数

- **panel** - [in] LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **mirror_x** - [in] Whether the panel will be mirrored about the x axis
- **mirror_y** - [in] Whether the panel will be mirrored about the y axis

#### 返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

```c
esp_err_t esp_lcd_panel_swap_xy(esp_lcd_panel_handle_t panel, bool swap_axes)
```

Swap/Exchange x and y axis.

**备注：** Combined with `esp_lcd_panel_mirror()`, one can realize screen rotation.

#### 参数

- **panel** - [in] LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **swap_axes** - [in] Whether to swap the x and y axis

#### 返回

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

```c
esp_err_t esp_lcd_panel_set_gap(esp_lcd_panel_handle_t panel, int x_gap, int y_gap)
```

Set extra gap in x and y axis.

The gap is the space (in pixels) between the left/top sides of the LCD panel and the first row/column respectively of the actual contents displayed.

**备注：** Setting a gap is useful when positioning or centering a frame that is smaller than the LCD.

#### 参数

- **panel** - [in] LCD panel handle, which is created by other factory API like `esp_lcd_new_panel_st7789()`
- **x_gap** - [in] Extra gap on x axis, in pixels
- **y_gap** - [in] Extra gap on y axis, in pixels

#### 返回

- ESP_OK on success
Esp_err_t esp_lcd_panel_invert_color(esp_lcd_panel_handle_t panel, bool invert_color_data)
Invert the color (bit-wise invert the color data line)

参数
• panel –[in] LCD panel handle, which is created by other factory API like esp_lcd_new_panel_st7789()
• invert_color_data –[in] Whether to invert the color data

返回
• ESP_OK on success

Esp_err_t esp_lcd_panel_disp_on_off(esp_lcd_panel_handle_t panel, bool on_off)
Turn on or off the display.

参数
• panel –[in] LCD panel handle, which is created by other factory API like esp_lcd_new_panel_st7789()
• on_off –[in] True to turn on display, False to turn off display

返回
• ESP_OK on success
• ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

Esp_err_t esp_lcd_panel_disp_off(esp_lcd_panel_handle_t panel, bool off)
Turn off the display.

参数
• panel –[in] LCD panel handle, which is created by other factory API like esp_lcd_new_panel_st7789()
• off –[in] Whether to turn off the screen

返回
• ESP_OK on success
• ESP_ERR_NOT_SUPPORTED if this function is not supported by the panel

Header File
• components/esp_lcd/include/esp_lcd_panel_rgb.h

Header File
• components/esp_lcd/include/esp_lcd_panel_vendor.h

Functions
Esp_err_t esp_lcd_new_panel_st7789( const esp_lcd_panel_io_handle_t io, const esp_lcd_panel_dev_config_t *panel_dev_config, esp_lcd_panel_handle_t *ret_panel )
Create LCD panel for model ST7789.

参数
• io –[in] LCD panel IO handle
• panel_dev_config –[in] general panel device configuration
• ret_panel –[out] Returned LCD panel handle

返回
• ESP_ERR_INVALID_ARG if parameter is invalid
• ESP_ERR_NO_MEM if out of memory
• ESP_OK on success

Esp_err_t esp_lcd_new_panel_nt35510( const esp_lcd_panel_io_handle_t io, const esp_lcd_panel_dev_config_t *panel_dev_config, esp_lcd_panel_handle_t *ret_panel )
Create LCD panel for model NT35510.
### Chapter 2. API 参考

**参数**
- `io` [in] LCD panel IO handle
- `panel_dev_config` [in] general panel device configuration
- `ret_panel` [out] Returned LCD panel handle

**返回**
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

```c
esp_err_t esp_lcd_new_panel_ssd1306(esp_lcd_panel_io_handle_t io, const esp_lcd_panel_dev_config_t *panel_dev_config, esp_lcd_panel_handle_t *ret_panel)
```

Create LCD panel for model SSD1306.

**参数**
- `io` [in] LCD panel IO handle
- `panel_dev_config` [in] general panel device configuration
- `ret_panel` [out] Returned LCD panel handle

**返回**
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

** Structures**

```c
struct esp_lcd_panel_dev_config_t
```

Configuration structure for panel device.

**Public Members**

```c
int reset_gpio_num
```

GPIO used to reset the LCD panel, set to -1 if it’s not used

```c
lcd_color_rgb_endian_t color_space
```

Deprecated:
- Set RGB color space, please use rgb_endian instead

```c
lcd_color_rgb_endian_t rgb_endian
```

Set RGB data endian: RGB or BGR

```c
unsigned int bits_per_pixel
```

Color depth, in bpp

```c
unsigned int reset_active_high
```

Setting this if the panel reset is high level active

```c
struct esp_lcd_panel_dev_config_t::[anonymous] flags
```

LCD panel config flags

```c
void *vendor_config
```

Vendor specific configuration, optional, left as NULL if not used
2.6.11 LED PWM 控制器

概述

LED 控制器 (LEDC) 主要用于控制 LED，也可产生 PWM 信号用于其他设备的控制。该控制器有 16 路通道，可以产生独立的波形来驱动 RGB LED 等设备。

LEDC 通道共有两组，分别为 8 路高速通道和 8 路低速通道。高速通道模式在硬件中实现，可以自动且无干扰地改变 PWM 占空比。低速通道模式下，PWM 占空比需要由软件中的驱动器改变。每组通道都可以使用不同的时钟源。

LED PWM 控制器可在无需 CPU 干预的情况下自动改变占空比，实现亮度和颜色渐变。

功能概览

设置 LEDC 通道在高速模式或低速模式下运行，需要进行如下配置：

1. 定时器配置 指定 PWM 信号的频率和占空比分辨率。
2. 通道配置 绑定定时器和输出 PWM 信号的 GPIO。
3. 改变 PWM 信号 输出 PWM 信号来驱动 LED。可通过软件控制或使用硬件渐变功能来改变 LED 的亮度。

另一个可选步骤是可以在渐变终端设置一个中断。

图 9: LED PWM 控制器 API 的关键配置

定时器配置 要设置定时器，可调用函数 ledc_timer_config()，并将包括如下配置参数的数据结构 ledc_timer_config_t 传递给该函数:

- 速度模式 ledc_mode_t
- 定时器索引 ledc_timer_t
- PWM 信号频率
- PWM 占空比分辨率
- 时钟源 ledc_clk_cfg_t
频率和占空比分辨率相互关联。PWM 频率越高，占空比分辨率越低，反之亦然。如果 API 不是用来改变 LED 亮度，而是用于其它目的，这种相互关系可能会很重要。更多信息详见频率和占空比分辨率支持范围一节。

时钟源同样可以限制 PWM 频率。选择的时钟源频率越高，可以配置的 PWM 频率上限就越高。

<table>
<thead>
<tr>
<th>时钟名称</th>
<th>时钟频率</th>
<th>速度模式</th>
<th>时钟功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>APB_CLK</td>
<td>80 MHz</td>
<td>高速 / 低速</td>
<td>/</td>
</tr>
<tr>
<td>REF_TICK</td>
<td>1 MHz</td>
<td>高速 / 低速</td>
<td>支持动态调频 (DFS) 功能</td>
</tr>
<tr>
<td>RTC8M_CLK</td>
<td>~8 MHz</td>
<td>低速</td>
<td>支持动态调频 (DFS) 功能，支持 Light-sleep 模式</td>
</tr>
</tbody>
</table>

备注：
1. 如果 ESP32 的定时器选用了 RTCxM_CLK 作为其时钟源，驱动会通过内部校准来得知这个时钟源的实频频率。这样确保了输出 PWM 信号频率的精确性。

通道配置 定时器设置好后，配置所需的通道（ledc_channel_t 之一）。配置通道需调用函数

ledc_channel_config()。

通过的配置与定时器设置类似，需向通道配置函数传递包括通道配置参数的结构体

ledc_channel_config_t。

此时，通道会按照ledc_channel_config_t 的配置开始运作，并在选定的 GPIO 上生成由定时器设置指定的频率和占空比的 PWM 信号。在通道运作过程中，可以随时通过调用函数 ledc_stop() 将其暂停。

改变 PWM 信号 通道开始运行，生成具有恒定占空比和频率的 PWM 信号之后，有几种方式可以改变该信号。驱动 LED 时，主要通过改变占空比来变化光线亮度。

以下两节介绍了如何使用软件和硬件改变占空比。如有需要，PWM 信号的频率也可更改，详见改变 PWM 频率 一节。

使用软件改变 PWM 占空比 调用函数

ledc_set_duty() 可以设置新的占空比。之后，调用函数

ledc_update_duty() 使新配置生效。要查看当前设置的占空比，可使用 _get_ 函数

ledc_get_duty()。

另外一种设置占空比和其他通道参数的方式是调用通道配置一节提到的函数

ledc_channel_config()。

传递给函数的占空比数值范围取决于选定的 duty_resolution，应为 0 至 (2 ** duty_resolution) - 1。例如，如选定的占空比分辨率为 10，则占空比的数值范围为 0 至 1023。此时分辨率为 ~0.1%。

使用硬件改变 PWM 占空比 LED PWM 控制器硬件可逐渐改变占空比的数值。要使用此功能，需用函数

ledc_fade_func_install() 使能渐变。之后用下列可用渐变函数之一配置：

• ledc_set_fade_with_time()
• ledc_set_fade_with_step()
• ledc_set_fade()

最后需调用 ledc_fade_start() 开启渐变。渐变可以在阻塞或非阻塞模式下运行，具体区别请查看

ledc_fade_mode_t，需要特别注意的是，不管在哪种模式下，一次渐变或单次占空比配置的指令生效都必须等到前一次渐变结束。由于 ESP32 的硬件限制，在渐变达到预先预期的占空比前想要中止本次渐变是不被支持的。

此外，在使能渐变后，每个通道都可以额外通过调用 ledc_cb_register() 注册一个回调函数用以获得渐变完成的事件通知。
如不需渐变和渐变中断，可用函数 `ledc_fade_func_uninstall()` 关闭。

**改变 PWM 频率**  LED PWM 控制器 API 有多种方式即时改变 PWM 频率：
- 通过调用函数 `ledc_set_freq()` 设置频率。可用函数 `ledc_get_freq()` 查看当前频率。
- 通过调用函数 `ledc_bind_channel_timer()` 将其他定时器绑定到该通道来改变频率和占空比分辨率。
- 通过调用函数 `ledc_channel_config()` 改变通道的定时器。

**控制 PWM 的更多方式** 有些较底层的定时器固定函数可用于更改 PWM 设置：
- `ledc_timer_set()`
- `ledc_timer_rst()`
- `ledc_timer_pause()`
- `ledc_timer_resume()`

前两个功能可通过函数 `ledc_channel_config()` 在后台运行，在定时器配置后启动。

**使用中断配置** LED PWM 控制器通道时，可在 `ledc_channel_config_t` 中选取参数 `ledc_intr_type_t`，在渐变完成时触发中断。

要注册处理程序来处理中断，可调用函数 `ledc_isr_register()`。

**LED PWM 控制器高速和低速模式**

高速模式的优点是平稳地改变定时器设置。也就是说，高速模式下如定时器设置改变，此变更会自动应用于定时器的下一次溢出中断。而更新低速定时器时，设置更应由软件显示触发。LED PWM 驱动的设置将在硬件层面被修改。比如在调用函数 `ledc_timer_config()` 或 `ledc_timer_set()` 时。

更多关于速度模式的详细信息请参阅 ESP32 技术参考手册 > LED PWM 控制器 (LEDC) [PDF]。

**频率和占空比分辨率支持范围**

LED PWM 控制器主要用于驱动 LED，该控制器 PWM 占空比设置的分辨率范围较广。比如，PWM 频率为 5 kHz 时，占空比分辨率最大可为 13 位。这意味着占空比可为 0 至 100% 之间的任意值，分辨率约为 0.012% (2^13 = 8192 LED 频率的离散电平)。然而，这些参数取决于为 LED PWM 控制器定时器定时的时钟信号。LED PWM 控制器为通道提供时钟（具体可参考定时器配置和 ESP32 技术参考手册 > LED PWM 定时器 (LEDC) [PDF])。

LED PWM 控制器可用于生成频率较高的信号。足以作为数码相机模组等其他设备提供时钟。此时，最大频率可为 40 MHz，占空比分辨率可为 1 位。也就是说，占空比固定为 50%，无法调整。

LED PWM 控制器 API 会在设定的频率和占空比分辨率超过 LED PWM 控制器硬件范围时报错。例如，试图将频率设置为 20 MHz、占空比分辨率设置为 3 位时，串行端口监视器上会报告如下错误：

```plaintext
E (196) ledc: requested frequency and duty resolution cannot be achieved, try reducing freq_hz or duty_resolution. div_param=128
```

此时，占空比分辨率或频率必须降低。比如，将占空比分辨率设置为 2 会解决这一问题，让占空比设置为 25% 的倍数，即 25%、50% 或 75%。

如设置的频率和占空比分辨率低于所支持的最低值，LED PWM 驱动器也会反映并报告，如：

```plaintext
E (196) ledc: requested frequency and duty resolution cannot be achieved, try increasing freq_hz or duty_resolution. div_param=128000000
```

占空比分辨率通常用 `ledc_timer_bit_t` 设置，范围是 10 至 15 位。如需较低的占空比分辨率（上至 10，下至 1），可直接输入相应数值。
Chapter 2. API 参考

应用实例

使用 LEDC 改变占空比和渐变控制的实例请参照 peripherals/ledc/ledc_fade。
使用 LEDC 基本实例请参照 peripherals/ledc/ledc_basic。

API 参考

Header File

- components/driver/include/driver/ledc.h

Functions

`esp_err_t ledc_channel_config(const ledc_channel_config_t *ledc_conf)`

LEDC channel configuration Configure LEDC channel with the given channel/output gpio_num/interrupt/source timer/frequency(Hz)/LEDC duty resolution.

参数 ledc_conf – Pointer of LEDC channel configure struct
返回
\[
\begin{align*}
&\text{ESP_OK Success} \\
&\text{ESP_ERR_INVALID_ARG Parameter error}
\end{align*}
\]

`esp_err_t ledc_timer_config(const ledc_timer_config_t *timer_conf)`

LEDC timer configuration Configure LEDC timer with the given source timer/frequency(Hz)/duty_resolution.

参数 timer_conf – Pointer of LEDC timer configure struct
返回
\[
\begin{align*}
&\text{ESP_OK Success} \\
&\text{ESP_ERR_INVALID_ARG Parameter error} \\
&\text{ESP_FAIL Cannot find a proper pre-dividernumberbase on the given frequency and the current duty_resolution.}
\end{align*}
\]

`esp_err_t ledc_update_duty(ledc_mode_t speed_mode, ledc_channel_t channel)`

LEDC update channel parameters.

备注：Call this function to activate the LEDC updated parameters. After ledc_set_duty, we need to call this function to update the settings. And the new LEDC parameters don’t take effect until the next PWM cycle.

备注：ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update

参数
\[
\begin{align*}
&\text{speed_mode } \text{– Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.} \\
&\text{channel } \text{– LEDC channel } (0 - \text{ LEDC_CHANNEL_MAX-1}), \text{ select from ledc_channel_t}
\end{align*}
\]
返回
\[
\begin{align*}
&\text{ESP_OK Success} \\
&\text{ESP_ERR_INVALID_ARG Parameter error}
\end{align*}
\]

`esp_err_t ledc_set_pin(int gpio_num, ledc_mode_t speed_mode, ledc_channel_t ledc_channel)`

Set LEDC output gpio.
Chapter 2. API参

备注：This function only routes the LEDC signal to GPIO through matrix, other LEDC resources initialization are not involved. Please use `ledc_channel_config()` instead to fully configure a LEDC channel.

参数
- `gpio_num` - The LEDC output gpio
- `speed_mode` - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `ledc_channel` - LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from `ledc_channel_t`

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

`esp_err_t ledc_stop (ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t idle_level)`
LEDC stop. Disable LEDC output, and set idle level.

参数
- `speed_mode` - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel` - LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from `ledc_channel_t`
- `idle_level` - Set output idle level after LEDC stops.

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

`esp_err_t ledc_set_freq (ledc_mode_t speed_mode, ledc_timer_t timer_num, uint32_t freq_hz)`
LEDC set channel frequency (Hz)

参数
- `speed_mode` - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `timer_num` - LEDC timer index (0-3), select from `ledc_timer_t`
- `freq_hz` - Set the LEDC frequency

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Can not find a proper pre-divider number base on the given frequency and the current duty_resolution.

`uint32_t ledc_get_freq (ledc_mode_t speed_mode, ledc_timer_t timer_num)`
LEDC get channel frequency (Hz)

参数
- `speed_mode` - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `timer_num` - LEDC timer index (0-3), select from `ledc_timer_t`

返回
- 0 error
- Others Current LEDC frequency

`esp_err_t ledc_set_duty_with_hpoint (ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t duty, uint32_t hpoint)`
LEDC set duty and hpoint value Only after calling `ledc_update_duty` will the duty update.

备注：`ledc_set_duty`, `ledc_set_duty_with_hpoint` and `ledc_update_duty` are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is
ledc_set_duty_and_update

**备注:** For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

### 参数
- **speed_mode** - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** - LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **duty** - Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution) - 1]
- **hpoint** - Set the LEDC hpoint value(max: 0xfffff)

### 返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

```c
int ledc_get_hpoint (ledc_mode_t speed_mode, ledc_channel_t channel)
```

LEDC get hpoint value, the counter value when the output is set high level.

### 参数
- **speed_mode** - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** - LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

### 返回
- LEDC_ERR_VAL if parameter error
- Others Current hpoint value of LEDC channel

```c
esp_err_t ledc_set_duty (ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t duty)
```

LEDC set duty This function do not change the hpoint value of this channel. if needed, please call ledc_set_duty_with_hpoint. only after calling ledc_update_duty will the duty update.

**备注:** ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update.

**备注:** For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

### 参数
- **speed_mode** - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** - LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **duty** - Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution) - 1]

### 返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

```c
uint32_t ledc_get_duty (ledc_mode_t speed_mode, ledc_channel_t channel)
```

LEDC get duty This function returns the duty at the present PWM cycle. You shouldn’t expect the function to return the new duty in the same cycle of calling ledc_update_duty, because duty update doesn’t take effect until the next cycle.
### 章节2. API

#### 参数

- **speed_mode** - 选择 LEDC 频率组与指定频率模式。注意不是所有目标支持高速模式。
- **channel** - LEDC 通道 (0 - LEDC_CHANNEL_MAX-1)，从 ledc_channel_t 选择。

#### 返回

- LEDC_ERR_DUTY if parameter error
- Others Current LEDC duty

**esp_err_t** `ledc_set_fade(speed_mode, channel, uint32_t duty, fade_direction, uint32_t step_num, uint32_t duty_cycle_num, uint32_t duty_scale)`

设置 LEDC 梯度。在该函数调用 lede_update_duty 函数之后，该函数才能生效。

**备注：** 为 ESP32，硬件不支持任何梯度更改，当在坡度操作运行时，该坡度操作将不会影响梯度。

#### 参数

- **speed_mode** - 选择 LEDC 频率组与指定频率模式。注意不是所有目标支持高速模式。
- **channel** - LEDC 通道 (0 - LEDC_CHANNEL_MAX-1)，从 ledc_channel_t 选择。
- **duty** - 设置梯度开始的载荷，设置范围是 [0, (2**duty_resolution) - 1]
- **fade_direction** - 设置梯度的方向
- **step_num** - 设置梯度的数量
- **duty_cycle_num** - 设置每个 LEDC 时钟周期的梯度持续时间
- **duty_scale** - 设置梯度变化幅度

#### 返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**esp_err_t** `ledc_isr_register(void (*fn)(void*), void* arg, int intr_alloc_flags, ledc_isr_handle_t *handle)`

注册 LEDC 中断处理器，处理器是一个中断服务函数。处理器将被附加到相同的 CPU 核上，此时这个函数正在运行。

**参数**

- **fn** - 中断处理器函数。
- **arg** - 用户提供的函数传递的参数。
- **intr_alloc_flags** - 用于分配中断的标志。一个或多个（ORed）ESP_INTR_FLAG_*值。更多信息，请参见 esp_intr_alloc.h。
- **handle** - 返回句柄的指针。如果非-NULL，句柄将返回给用户。

#### 返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Function pointer error

**esp_err_t** `ledc_timer_set(speed_mode, timer_sel, uint32_t clock_divider, uint32_t duty_resolution, ledc_clk_src_t clk_src)`

配置 LEDC 设置。

**参数**

- **speed_mode** - 选择 LEDC 频率组与指定频率模式。注意不是所有目标支持高速模式。
- **timer_sel** - 时钟索引 (0-3)，有 4 个时钟在 LEDC 模块
- **clock_divider** - 时钟分频值，时钟被分割成所选的时钟源
• **duty_resolution** – Resolution of duty setting in number of bits. The range of duty values is \([0, (2^{\text{duty_resolution}})]\)
• **clk_src** – Select LEDC source clock.

返回
• (-1) Parameter error
• Other Current LEDC duty

`esp_err_t ledc_timer_rst (ledc_mode_t speed_mode, ledc_timer_t timer_sel)`
Reset LEDC timer.

参数
• **speed_mode** – Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
• **timer_sel** – LEDC timer index (0-3), select from ledc_timer_t

返回
• ESP_ERR_INVALID_ARG Parameter error
• ESP_OK Success

`esp_err_t ledc_timer_pause (ledc_mode_t speed_mode, ledc_timer_t timer_sel)`
Pause LEDC timer counter.

参数
• **speed_mode** – Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
• **timer_sel** – LEDC timer index (0-3), select from ledc_timer_t

返回
• ESP_ERR_INVALID_ARG Parameter error
• ESP_OK Success

`esp_err_t ledc_timer_resume (ledc_mode_t speed_mode, ledc_timer_t timer_sel)`
Resume LEDC timer.

参数
• **speed_mode** – Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
• **timer_sel** – LEDC timer index (0-3), select from ledc_timer_t

返回
• ESP_ERR_INVALID_ARG Parameter error
• ESP_OK Success

`esp_err_t ledc_bind_channel_timer (ledc_mode_t speed_mode, ledc_channel_t channel, ledc_timer_t timer_sel)`
Bind LEDC channel with the selected timer.

参数
• **speed_mode** – Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
• **channel** – LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
• **timer_sel** – LEDC timer index (0-3), select from ledc_timer_t

返回
• ESP_ERR_INVALID_ARG Parameter error
• ESP_OK Success

`esp_err_t ledc_set_fade_with_step (ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t target_duty, uint32_t scale, uint32_t cycle_num)`
Set LEDC fade function.

备注: Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.
Chapter 2. API

备注：ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_fade_step_and_start

备注：For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数
- **speed_mode** - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** - LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **target_duty** - Target duty of fading [0, (2**duty_resolution) - 1]
- **scale** - Controls the increase or decrease step scale.
- **cycle_num** - increase or decrease the duty every cycle_num cycles

返回
- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

```c
esp_err_t ledc_set_fade_with_time(ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t target_duty, int max_fade_time_ms)
```

Set LEDC fade function, with a limited time.

备注：Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.

备注：ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_fade_step_and_start

备注：For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数
- **speed_mode** - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** - LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **target_duty** - Target duty of fading [0, (2**duty_resolution) - 1]
- **max_fade_time_ms** - The maximum time of the fading (ms).

返回
- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

```c
esp_err_t ledc_fade_func_install (int intr_alloc_flags)
```

Install LEDC fade function. This function will occupy interrupt of LEDC module.
参数 `intr_alloc_flags` — Flags used to allocate the interrupt. One or multiple (ORred) `ESP_INTR_FLAG_*` values. See `esp_intr_alloc.h` for more info.

返回
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function already installed.

`void ledc_fade_func_uninstall (void)`
Uninstall LEDC fade function.

`esp_err_t ledc_fade_start (ledc_mode_t speed_mode, ledc_channel_t channel, ledc_fade_mode_t fade_mode)`
Start LEDC fading.

备注： Call `ledc_fade_func_install()` once before calling this function. Call this API right after `ledc_set_fade_with_time` or `ledc_set_fade_with_step` before to start fading.

备注： Starting fade operation with this API is not thread-safe, use with care.

备注： For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数
- `speed_mode` — Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel` — LEDC channel number
- `fade_mode` — Whether to block until fading done. See `ledc_types.h ledc_fade_mode_t` for more info. Note that this function will not return until fading to the target duty if `LEDC_FADE_WAIT_DONE` mode is selected.

返回
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_ERR_INVALID_ARG Parameter error.

`esp_err_t ledc_set_duty_and_update (ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t duty, uint32_t hpoint)`
A thread-safe API to set duty for LEDC channel and return when duty updated.

备注： For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

参数
- `speed_mode` — Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel` — LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from `ledc_channel_t`
- `duty` — Set the LEDC duty, the range of duty setting is `[0, (2**duty_resolution) - 1]`
- `hpoint` — Set the LEDC hpoint value(max: 0xfffff)

`esp_err_t ledc_set_fade_time_and_start (ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t target_duty, uint32_t max_fade_time_ms, ledc_fade_mode_t fade_mode)`
A thread-safe API to set and start LEDC fade function, with a limited time.
Chapter 2. API 参考

Call `ledc_fade_func_install()` once, before calling this function.

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

### 参数
- `speed_mode` - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel` - LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from `ledc_channel_t`
- `target_duty` - Target duty of fading [0, (2**duty_resolution) - 1]
- `max fade time ms` - The maximum time of the fading ( ms ).
- `fade_mode` - Choose blocking or non-blocking mode

### 返回
- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

```c
esp_err_t ledc_set_fade_step_and_start (ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t target_duty, uint32_t scale, uint32_t cycle_num, ledc_fade_mode_t fade_mode)
```

A thread-safe API to set and start LEDC fade function.

Call `ledc_fade_func_install()` once before calling this function.

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

### 参数
- `speed_mode` - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel` - LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from `ledc_channel_t`
- `target_duty` - Target duty of fading [0, (2**duty_resolution) - 1]
- `scale` - Controls the increase or decrease step scale.
- `cycle_num` - Increase or decrease the duty every cycle_num cycles
- `fade_mode` - Choose blocking or non-blocking mode

### 返回
- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

```c
esp_err_t ledc_cb_register (ledc_mode_t speed_mode, ledc_channel_t channel, ledc_cbs_t *cbs, void *user_arg)
```

LEDC callback registration function.

The callback is called from an ISR, it must never attempt to block, and any FreeRTOS API called must be ISR capable.
参数

- **speed_mode** - Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** - LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **cbs** - Group of LEDC callback functions
- **user_arg** - user registered data for the callback function

返回

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

**Structures**

struct **ledc_channel_config_t**

Configuration parameters of LEDC channel for ledc_channel_config function.

**Public Members**

```c
int gpio_num
```

the LEDC output gpio_num, if you want to use gpio16, gpio_num = 16

```c
ledc_mode_t speed_mode
```

LEDC speed speed_mode, high-speed mode or low-speed mode

```c
ledc_channel_t channel
```

LEDC channel (0 - 7)

```c
ledc_intr_type_t intr_type
```

configure interrupt, Fade interrupt enable or Fade interrupt disable

```c
ledc_timer_t timer_sel
```

Select the timer source of channel (0 - 3)

```c
uint32_t duty
```

LEDC channel duty, the range of duty setting is \([0, (2^{*}duty\_resolution)]\)

```c
int hpoint
```

LEDC channel hpoint value, the max value is 0xffffffff

```c
unsigned int output_invert
```

Enable (1) or disable (0) gpio output invert

```c
struct ledc_channel_config_t::[anonymous] flags
```

LEDC flags

struct **ledc_timer_config_t**

Configuration parameters of LEDC Timer timer for ledc_timer_config function.
Public Members

`ledc_mode_t speed_mode`
LEDC speed speed_mode, high-speed mode or low-speed mode

`ledc_timer_bit_t duty_resolution`
LEDC channel duty resolution

`ledc_timer_t timer_num`
The timer source of channel (0 - 3)

`uint32_t freq_hz`
LEDC timer frequency (Hz)

`ledc_clk_cfg_t clk_cfg`
Configure LEDC source clock from ledc_clk_cfg_t. Note that LEDC_USE_RTC8M_CLK and LEDC_USE_XTAL_CLK are non-timer-specific clock sources. You cannot have one LEDC timer uses RTC8M_CLK as the clock source and have another LEDC timer uses XTAL_CLK as its clock source. All chips except esp32 and esp32s2 do not have timer-specific clock sources, which means clock source for all timers must be the same one.

struct `ledc_cb_param_t`
LEDC callback parameter.

Public Members

`ledc_cb_event_t event`
Event name

`uint32_t speed_mode`
Speed mode of the LEDC channel group

`uint32_t channel`
LEDC channel (0 - LEDC_CHANNEL_MAX-1)

`uint32_t duty`
LEDC current duty of the channel, the range of duty is [0, (2**duty_resolution) - 1]

struct `ledc_cbs_t`
Group of supported LEDC callbacks.

备注: The callbacks are all running under ISR environment

Public Members

`ledc_cb_t fade_cb`
LEDC fade_end callback function
Chapter 2. API

Macros

LEDC_APB_CLK_HZ
Frequency of one of the LEDC peripheral clock sources, APB_CLK.

备注: This macro should have no use in your application, we keep it here only for backward compatible

LEDC_REF_CLK_HZ
Frequency of one of the LEDC peripheral clock sources, REF_TICK.

备注: This macro should have no use in your application, we keep it here only for backward compatible

LEDC_ERR_DUTY

LEDC_ERR_VAL

Type Definitions

typedef intr_handle_t ledc_isr_handle_t

typedef bool (*ledc_cb_t)(const ledc_cb_param_t *param, void *user_arg)
Type of LEDC event callback.

Param param LEDC callback parameter
Param user_arg User registered data
Return Whether a high priority task has been waken up by this function

Enumerations

enum ledc_cb_event_t
LEDC callback event type.

Values:

enumerator LEDC_FADE_END_EVT
LEDC fade end event

Header File

• components/hal/include/hal/ledc_types.h

Enumerations

enum ledc_mode_t

Values:

enumerator LEDC_HIGH_SPEED_MODE
LEDC high speed speed_mode

enumerator LEDC_LOW_SPEED_MODE
LEDC low speed speed_mode
enumerator **LEDC_SPEED_MODE_MAX**
   LEDC speed limit

enum **ledc_intr_type_t**

*Values:*

enumerator **LEDC_INTR_DISABLE**
   Disable LEDC interrupt

enumerator **LEDC_INTR_FADE_END**
   Enable LEDC interrupt

enumerator **LEDC_INTR_MAX**

enum **ledc_duty_direction_t**

*Values:*

enumerator **LEDC_DUTY_DIR_DECREASE**
   LEDC duty decrease direction

enumerator **LEDC_DUTY_DIR_INCREASE**
   LEDC duty increase direction

enumerator **LEDC_DUTY_DIR_MAX**

enum **ledc_slow_clk_sel_t**

*Values:*

enumerator **LEDC_SLOW_CLK_RTC8M**
   LEDC low speed timer clock source is 8MHz RTC clock

enumerator **LEDC_SLOW_CLK_APB**
   LEDC low speed timer clock source is 80MHz APB clock

enum **ledc_clk_cfg_t**

In theory, the following enumeration shall be placed in LEDC driver’s header. However, as the next enumeration, **ledc_clk_src_t**, makes the use of some of these values and to avoid mutual inclusion of the headers, we must define it here.

*Values:*

enumerator **LEDC_AUTO_CLK**
   The driver will automatically select the source clock based on the giving resolution and duty parameter when init the timer

enumerator **LEDC_USE_APB_CLK**
   LEDC timer select APB clock as source clock

enumerator **LEDC_USE_RTC8M_CLK**
   LEDC timer select RTC8M_CLK as source clock. Only for low speed channels and this parameter must be the same for all low speed channels
Chapter 2. API

**enumerator** `LEDC_USE_REF_TICK`
- LEDC timer select REF_TICK clock as source clock

**enum** `ledc_clk_src_t`
- **enumerator** `LEDC_REF_TICK`
  - LEDC timer clock divided from reference tick (1Mhz)
- **enumerator** `LEDC_APB_CLK`
  - LEDC timer clock divided from APB clock (80Mhz)
- **enumerator** `LEDC_SCLK`
  - Selecting this value for LEDC_TICK_SEL_TIMER let the hardware take its source clock from LEDC_APB_CLK_SEL

**enum** `ledc_timer_t`
- **enumerator** `LEDC_TIMER_0`
  - LEDC timer 0
- **enumerator** `LEDC_TIMER_1`
  - LEDC timer 1
- **enumerator** `LEDC_TIMER_2`
  - LEDC timer 2
- **enumerator** `LEDC_TIMER_3`
  - LEDC timer 3
- **enumerator** `LEDC_TIMER_MAX`

**enum** `ledc_channel_t`
- **enumerator** `LEDC_CHANNEL_0`
  - LEDC channel 0
- **enumerator** `LEDC_CHANNEL_1`
  - LEDC channel 1
- **enumerator** `LEDC_CHANNEL_2`
  - LEDC channel 2
- **enumerator** `LEDC_CHANNEL_3`
  - LEDC channel 3
enumerator LEDC_CHANNEL_4
   LEDC channel 4

enumerator LEDC_CHANNEL_5
   LEDC channel 5

enumerator LEDC_CHANNEL_6
   LEDC channel 6

enumerator LEDC_CHANNEL_7
   LEDC channel 7

enumerator LEDC_CHANNEL_MAX

enum ledc_timer_bit_t

   Values:

enumerator LEDC_TIMER_1_BIT
   LEDC PWM duty resolution of 1 bits

enumerator LEDC_TIMER_2_BIT
   LEDC PWM duty resolution of 2 bits

enumerator LEDC_TIMER_3_BIT
   LEDC PWM duty resolution of 3 bits

enumerator LEDC_TIMER_4_BIT
   LEDC PWM duty resolution of 4 bits

enumerator LEDC_TIMER_5_BIT
   LEDC PWM duty resolution of 5 bits

enumerator LEDC_TIMER_6_BIT
   LEDC PWM duty resolution of 6 bits

enumerator LEDC_TIMER_7_BIT
   LEDC PWM duty resolution of 7 bits

enumerator LEDC_TIMER_8_BIT
   LEDC PWM duty resolution of 8 bits

enumerator LEDC_TIMER_9_BIT
   LEDC PWM duty resolution of 9 bits

enumerator LEDC_TIMER_10_BIT
   LEDC PWM duty resolution of 10 bits

enumerator LEDC_TIMER_11_BIT
   LEDC PWM duty resolution of 11 bits
Chapter 2. API Reference

**enum** `LEDC_TIMER_12_BIT`  
LEDC PWM duty resolution of 12 bits

**enum** `LEDC_TIMER_13_BIT`  
LEDC PWM duty resolution of 13 bits

**enum** `LEDC_TIMER_14_BIT`  
LEDC PWM duty resolution of 14 bits

**enum** `LEDC_TIMER_15_BIT`  
LEDC PWM duty resolution of 15 bits

**enum** `LEDC_TIMER_16_BIT`  
LEDC PWM duty resolution of 16 bits

**enum** `LEDC_TIMER_17_BIT`  
LEDC PWM duty resolution of 17 bits

**enum** `LEDC_TIMER_18_BIT`  
LEDC PWM duty resolution of 18 bits

**enum** `LEDC_TIMER_19_BIT`  
LEDC PWM duty resolution of 19 bits

**enum** `LEDC_TIMER_20_BIT`  
LEDC PWM duty resolution of 20 bits

**enum** `LEDC_TIMER_BIT_MAX`  
LEDC PWM duty resolution of 20 bits

**enum** `ledc_fade_mode_t`  
*Values:*

**enum** `LEDC_FADE_NO_WAIT`  
LEDC fade function will return immediately

**enum** `LEDC_FADE_WAIT_DONE`  
LEDC fade function will block until fading to the target duty

**enum** `LEDC_FADE_MAX`  

### 2.6.12 Motor Control Pulse Width Modulator (MCPWM)

The MCPWM peripheral is a versatile PWM generator, which contains various submodules to make it a key element in power electronic applications like motor control, digital power and so on. Typically, the MCPWM peripheral can be used in the following scenarios:

- Digital motor control, e.g. brushed/brushless DC motor, RC servo motor
- Switch mode based digital power conversion
- Power DAC, where the duty cycle is equivalent to a DAC analog value
- Calculate external pulse width, and convert it into other analog value like speed, distance
• **Generate Space Vector PWM (SVPWM) signals for Field Oriented Control (FOC)**

The main submodules are listed in the following diagram:

![Diagram of MCPWM submodules](image)

**图 10: MCPWM Overview**

- **MCPWM Timer**: The time base of the final PWM signal, it also determines the event timing of other sub-modules.
- **MCPWM Operator**: The key module that is responsible for generating the PWM waveforms. It consists of other submodules, like comparator, PWM generator, dead-time and carrier modulator.
- **MCPWM Comparator**: The compare module takes the time-base count value as input, and continuously compare to the threshold value that configured by user. When the time-base counter is equal to any of the threshold value, an compare event will be generated and the MCPWM generator can update its level accordingly.
- **MCPWM Generator**: One MCPWM generator can generate a pair of PWM waves, complementarily or independently, based on various events triggered from other submodules like MCPWM Timer, MCPWM Comparator.
- **MCPWM Fault**: The fault module is used to detect the fault condition from outside, mainly via GPIO matrix. Once the fault signal is active, MCPWM Operator will force all the generators into a predefined state, to protect the system from damage.
- **MCPWM Sync**: The sync module is used to synchronize the MCPWM timers, so that the final PWM signals generated by different MCPWM generators can have a fixed phase difference. The sync signal can be routed from GPIO matrix or from MCPWM Timer event.
- **Dead Time**: This submodule is used to insert extra delay to the existing PWM edges that generated in the previous steps.
- **Carrier Modulation**: The carrier submodule allows a high-frequency carrier signal to modulate the PWM waveforms generated by the generator and dead time submodules. This capability is mandatory if you need pulse transformer-based gate drivers to control the power switching elements.
- **Brake**: MCPWM operator can set how to brake the generators when particular fault is detected. We can shut down the PWM output immediately or regulate the PWM output cycle by cycle, depends on how critical the fault is.
- **MCPWM Capture**: This is a standalone submodule which can work even without the above MCPWM operators. The capture consists one dedicated timer and several independent channels. Each channel is connected to the GPIO, a pulse on the GPIO will trigger the capture timer to store the time-base count value and then notify the user by interrupt. Using this feature, we can measure a pulse width precisely. What’s more, the capture timer can also be synchronized by the MCPWM Sync submodule.

**Functional Overview**

Description of the MCPWM functionality is divided into the following sections:

- **Resource Allocation and Initialization** - covers how to allocate various MCPWM objects, like timers, operators, comparators, generators and so on. These objects are the basis of the following IO setting and control functions.
• **Timer Operations and Events** - describes control functions and event callbacks that supported by the MCPWM timer.

• **Comparator Operations and Events** - describes control functions and event callbacks that supported by the MCPWM comparator.

• **Generator Actions on Events** - describes how to set actions for MCPWM generators on particular events that generated by the MCPWM timer and comparators.

• **Classical PWM Waveforms and Generator Configurations** - demonstrates some classical PWM waveforms that can be achieved by configuring generator actions.

• **Dead Time** - describes how to set dead time for MCPWM generators.

• **Classical PWM Waveforms and Dead Time Configurations** - demonstrates some classical PWM waveforms that can be achieved by configuring dead time.

• **Carrier Modulation** - describes how to set modulate a high frequency onto the final PWM waveforms.

• **Faults and Brake Actions** - describes how to set brake actions for MCPWM operators on particular fault event.

• **Generator Force Actions** - describes how to control the generator output level asynchronously in a forceful way.

• **Synchronization** - describes how to synchronize the MCPWM timers and get a fixed phase difference between the generated PWM signals.

• **Capture** - describes how to use the MCPWM capture module to measure the pulse width of a signal.

• **Power Management** - describes how different source clock will affect power consumption.

• **IRAM Safe** - describes tips on how to make the RMT interrupt work better along with a disabled cache.

• **Thread Safety** - lists which APIs are guaranteed to be thread safe by the driver.

• **Kconfig Options** - lists the supported Kconfig options that can bring different effects to the driver.

**Resource Allocation and Initialization** As displayed in the diagram above, the MCPWM peripheral consists of several submodules. Each submodule has its own resource allocation, which is described in the following sections.

**MCPWM Timers** You can allocate a MCPWM timer object by calling `mcpwm_new_timer()` function, with a configuration structure `mcpwm_timer_config_t` as the parameter. The configuration structure is defined as:

- `mcpwm_timer_config_t::group_id` specifies the MCPWM group ID. The ID should belong to [0, `SOC_MCPWM_GROUPS` - 1] range. Please note, timers located in different groups are totally independent.

- `mcpwm_timer_config_t::clk_src` sets the clock source of the timer.

- `mcpwm_timer_config_t::resolution_hz` set the expected resolution of the timer, the driver internally will set a proper divider based on the clock source and the resolution.

- `mcpwm_timer_config_t::count_mode` set the count mode of the timer.

- `mcpwm_timer_config_t::period_ticks` sets the period of the timer, in ticks (the tick resolution is set in the `mcpwm_timer_config_t::resolution_hz`).

- `mcpwm_timer_config_t::update_period_on_empty` sets whether to update the period value when the timer counts to zero.

- `mcpwm_timer_config_t::update_period_on_sync` sets whether to update the period value when the timer takes a sync signal.

The `mcpwm_new_timer()` will return a pointer to the allocated timer object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no more free timers in the MCPWM group, this function will return `ESP_ERR_NOT_FOUND` error.\(^1\)

On the contrary, calling `mcpwm_del_timer()` function will free the allocated timer object.

**MCPWM Operators** You can allocate a MCPWM operator object by calling `mcpwm_new_operator()` function, with a configuration structure `mcpwm_operator_config_t` as the parameter. The configuration structure is defined as:

- `mcpwm_operator_config_t::group_id` specifies the MCPWM group ID. The ID should belong to [0, `SOC_MCPWM_GROUPS` - 1] range. Please note, operators located in different groups are totally independent.

\(^1\) Different ESP chip series might have different number of MCPWM resources (e.g. groups, timers, comparators, operators, generators and so on). Please refer to the [TRM] for details. The driver won’t forbid you from applying for more MCPWM resources, but it will return error when there’s no hardware resources available. Please always check the return value when doing Resource Allocation.
• `mcpwm_operator_config_t::update_gen_action_on_tez` sets whether to update the generator action when the timer counts to zero. Here and below, the timer refers to the one that is connected to the operator by `mcpwm_operator_connect_timer()`.
• `mcpwm_operator_config_t::update_gen_action_on_tep` sets whether to update the generator action when the timer counts to peak.
• `mcpwm_operator_config_t::update_gen_action_on_sync` sets whether to update the generator action when the timer takes a sync signal.
• `mcpwm_operator_config_t::update_dead_time_on_tez` sets whether to update the dead time when the timer counts to zero.
• `mcpwm_operator_config_t::update_dead_time_on_tep` sets whether to update the dead time when the timer counts to peak.
• `mcpwm_operator_config_t::update_dead_time_on_sync` sets whether to update the dead time when the timer takes a sync signal.

The `mcpwm_new_operator()` function will return a pointer to the allocated operator object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no more free operators in the MCPWM group, this function will return `ESP_ERR_NOT_FOUND` error.

On the contrary, calling `mcpwm_del_operator()` function will free the allocated operator object.

**MCPWM Comparators** You can allocate a MCPWM comparator object by calling `mcpwm_new_comparator()` function, with a MCPWM operator handle and configuration structure `mcpwm_comparator_config_t` as the parameter. The operator handle is created by `mcpwm_new_operator()`(). The configuration structure is defined as:

• `mcpwm_comparator_config_t::update_cmp_on_tez` sets whether to update the compare threshold when the timer counts to zero.
• `mcpwm_comparator_config_t::update_cmp_on_tep` sets whether to update the compare threshold when the timer counts to peak.
• `mcpwm_comparator_config_t::update_cmp_on_sync` sets whether to update the compare threshold when the timer takes a sync signal.

The `mcpwm_new_comparator()` function will return a pointer to the allocated comparator object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no more free comparators in the MCPWM operator, this function will return `ESP_ERR_NOT_FOUND` error.

On the contrary, calling `mcpwm_del_comparator()` function will free the allocated comparator object.

**MCPWM Generators** You can allocate a MCPWM generator object by calling `mcpwm_new_generator()` function, with a MCPWM operator handle and configuration structure `mcpwm_generator_config_t` as the parameter. The operator handle is created by `mcpwm_new_operator()`(). The configuration structure is defined as:

• `mcpwm_generator_config_t::gen_gpio_num` sets the GPIO number used by the generator.
• `mcpwm_generator_config_t::invert_pwm` sets whether to invert the PWM signal.
• `mcpwm_generator_config_t::io_loop_back` sets whether to enable the loop back mode. It is for debugging purposes only. It enables both the GPIO’s input and output ability through the GPIO matrix peripheral.

The `mcpwm_new_generator()` function will return a pointer to the allocated generator object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no more free generators in the MCPWM operator, this function will return `ESP_ERR_NOT_FOUND` error.

On the contrary, calling `mcpwm_del_generator()` function will free the allocated generator object.

**MCPWM Faults** There are two types of faults: A fault signal reflected from the GPIO and a fault generated by software. To allocate a GPIO fault object, you can call `mcpwm_new_gpio_fault()` function, with configuration structure `mcpwm_gpio_fault_config_t` as the parameter. The configuration structure is defined as:
Chapter 2. API

- `mcpwm_gpio_fault_config_t::group_id` sets the MCPWM group ID. The ID should belong to `[0, SOC_MCPWM_GROUPS - 1]` range. Please note, GPIO fault located in different groups are totally independent, i.e. GPIO fault in group 0 cannot be detected by the operator in group 1.
- `mcpwm_gpio_fault_config_t::gpio_num` sets the GPIO number used by the fault.
- `mcpwm_gpio_fault_config_t::active_level` sets the active level of the fault signal.
- `mcpwm_gpio_fault_config_t::pull_up` and `mcpwm_gpio_fault_config_t::pull_down` set whether to pull up and/or pull down the GPIO internally.
- `mcpwm_gpio_fault_config_t::io_loop_back` sets whether to enable the loop back mode. It is for debugging purposes only. It enables both the GPIO’s input and output ability through the GPIO matrix peripheral.

The `mcpwm_new_gpio_fault()` will return a pointer to the allocated fault object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no more free GPIO faults in the MCPWM group, this function will return `ESP_ERR_NOT_FOUND` error.

Software fault object can be used to trigger a fault by calling a function `mcpwm_soft_fault_activate()` instead of waiting for a real fault signal on the GPIO. A software fault object can be allocated by calling `mcpwm_new_soft_fault()` function, with configuration structure `mcpwm_soft_fault_config_t` as the parameter. Currently this configuration structure is left for future purpose. `mcpwm_new_soft_fault()` function will return a pointer to the allocated fault object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no memory left for the fault object, this function will return `ESP_ERR_NO_MEM` error. Although the software fault and GPIO fault are of different types, but the returned fault handle is of the same type.

On the contrary, calling `mcpwm_del_fault()` function will free the allocated fault object, this function works for both software and GPIO fault.

**MCPWM Sync Sources** The sync source is what can be used to synchronize the MCPWM timer and MCPWM capture timer. There’re three types of sync sources: A sync source reflected from the GPIO, a sync source generated by software and a sync source generated by MCPWM timer event.

To allocate a GPIO sync source, you can call `mcpwm_new_gpio_sync_src()` function, with configuration structure `mcpwm_gpio_sync_src_config_t` as the parameter. The configuration structure is defined as:

- `mcpwm_gpio_sync_src_config_t::group_id` sets the MCPWM group ID. The ID should belong to `[0, SOC_MCPWM_GROUPS - 1]` range. Please note, GPIO sync source located in different groups are totally independent, i.e. GPIO sync source in group 0 cannot be detected by the timers in group 1.
- `mcpwm_gpio_sync_src_config_t::gpio_num` sets the GPIO number used by the sync source.
- `mcpwm_gpio_sync_src_config_t::active_neg` sets whether the sync signal is active on falling edge.
- `mcpwm_gpio_sync_src_config_t::pull_up` and `mcpwm_gpio_sync_src_config_t::pull_down` set whether to pull up and/or pull down the GPIO internally.
- `mcpwm_gpio_sync_src_config_t::io_loop_back` sets whether to enable the loop back mode. It is for debugging purposes only. It enables both the GPIO’s input and output ability through the GPIO matrix peripheral.

The `mcpwm_new_gpio_sync_src()` will return a pointer to the allocated sync source object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no more free GPIO sync sources in the MCPWM group, this function will return `ESP_ERR_NOT_FOUND` error.

To allocate a Timer event sync source, you can call `mcpwm_new_timer_sync_src()` function, with configuration structure `mcpwm_timer_sync_src_config_t` as the parameter. The configuration structure is defined as:

- `mcpwm_timer_sync_src_config_t::timer_event` specifies on what timer event to generate the sync signal.
- `mcpwm_timer_sync_src_config_t::propagate_input_sync` sets whether to propagate the input sync signal (i.e. the input sync signal will be routed to its sync output).

The `mcpwm_new_timer_sync_src()` will return a pointer to the allocated sync source object if the allocation succeeds. Otherwise, it will return error code. Specifically, if a sync source has been allocated from the same timer before, this function will return `ESP_ERR_INVALID_STATE` error.
Last but not least, to allocate a software sync source, you can call \texttt{mcpwm_new_soft_sync_src()} function, with configuration structure \texttt{mcpwm_soft_sync_config_t} as the parameter. Currently this configuration structure is left for future purpose. \texttt{mcpwm_new_soft_sync_src()} will return a pointer to the allocated sync source object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no memory left for the sync source object, this function will return \texttt{ESP\_ERR\_NO\_MEM} error. Please note, to make a software sync source take effect, don’t forget to call \texttt{mcpwm_soft_sync_activate()}.

On the contrary, calling \texttt{mcpwm_del_sync_src()} function will free the allocated sync source object, this function works for all types of sync sources.

**MCPWM Capture Timer and Channels** The MCPWM group has a dedicated timer which is used to capture the timestamp when specific event occurred. The capture timer is connected with several independent channels, each channel is assigned with a GPIO.

To allocate a capture timer, you can call \texttt{mcpwm_new_capture_timer()} function, with configuration structure \texttt{mcpwm_capture_timer_config_t} as the parameter. The configuration structure is defined as:

- \texttt{mcpwm_capture_timer_config_t::group_id} sets the MCPWM group ID. The ID should belong to \([0, SOC\_MCPWM\_GROUPS - 1]\) range.
- \texttt{mcpwm_capture_timer_config_t::clk_src} sets the clock source of the capture timer.

The \texttt{mcpwm_new_capture_timer()} will return a pointer to the allocated capture timer object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no free capture timer left in the MCPWM group, this function will return \texttt{ESP\_ERR\_NOT\_FOUND} error.

Next, to allocate a capture channel, you can call \texttt{mcpwm_new_capture_channel()} function, with a capture timer handle and configuration structure \texttt{mcpwm_capture_channel_config_t} as the parameter. The configuration structure is defined as:

- \texttt{mcpwm_capture_channel_config_t::gpio_num} sets the GPIO number used by the capture channel.
- \texttt{mcpwm_capture_channel_config_t::prescale} sets the prescaler of the input signal.
- \texttt{mcpwm_capture_channel_config_t::pos_edge} and \texttt{mcpwm_capture_channel_config_t::neg_edge} set whether to capture on the positive and/or negative edge of the input signal.
- \texttt{mcpwm_capture_channel_config_t::pull_up} and \texttt{mcpwm_capture_channel_config_t::pull_down} set whether to pull up and/or pull down the GPIO internally.
- \texttt{mcpwm_capture_channel_config_t::invert_cap_signal} sets whether to invert the capture signal.
- \texttt{mcpwm_capture_channel_config_t::io_loop_back} sets whether to enable the loop back mode. It is for debugging purposes only. It enables both the GPIO’s input and output ability through the GPIO matrix peripheral.

The \texttt{mcpwm_new_capture_channel()} will return a pointer to the allocated capture channel object if the allocation succeeds. Otherwise, it will return error code. Specifically, when there are no free capture channel left in the capture timer, this function will return \texttt{ESP\_ERR\_NOT\_FOUND} error.

On the contrary, calling \texttt{mcpwm_del_capture_channel()} and \texttt{mcpwm_del_capture_timer()} function will free the allocated capture channel and timer object accordingly.

**Timer Operations and Events**

**Register Event Callbacks** The MCPWM timer can generate different events at runtime. If you have some function that should be called when particular event happens, you should hook your function to the interrupt service routine by calling \texttt{mcpwm_timer_register_event_callbacks()} function prototype is declared in \texttt{mcpwm_timer_event_cb_t}. All supported event callbacks are listed in the \texttt{mcpwm_timer_event_callbacks_t}:

- \texttt{mcpwm_timer_event_callbacks_t::on_full} sets callback function for timer when it counts to peak value.
- \texttt{mcpwm_timer_event_callbacks_t::on_empty} sets callback function for timer when it counts to zero.
Chapter 2. API

**mcpwm_timer_event_callbacks_t::on_stop** sets callback function for timer when it is stopped.

The callback functions above are called within the ISR context, so they should **not** attempt to block (e.g., make sure that only FreeRTOS APIs with **ISR** suffix is called within the function).

The parameter **user_data** of **mcpwm_timer_register_event_callbacks()** function is used to save user’s own context, it will be passed to each callback function directly.

This function will lazy install interrupt service for the MCPWM timer without enabling it. It is only allowed to be called before before **mcpwm_timer_enable()**, otherwise the **ESP_ERR_INVALID_STATE** error will be returned. See also *Enable and Disable timer* for more information.

**Enable and Disable Timer** Before doing IO control to the timer, user needs to enable the timer first, by calling **mcpwm_timer_enable()**. Internally, this function will:

- switch the timer state from **init** to **enable**.
- enable the interrupt service if it has been lazy installed by **mcpwm_timer_register_event_callbacks()**.
- acquire a proper power management lock if a specific clock source (e.g. PLL_160M clock) is selected. See also *Power management* for more information.

On the contrary, calling **mcpwm_timer_disable()** will put the timer driver back to **init** state, disable the interrupts service and release the power management lock.

**Start and Stop Timer** The basic IO operation of a timer is to start and stop. Calling **mcpwm_timer_start_stop()** with different **mcpwm_timer_start_stop_cmd_t** commands can start the timer immediately or stop the timer at a specific event. What’s more, you can even start the timer for only one round, that means, the timer will count to peak value or zero, and then stop itself.

**Connect Timer with Operator** The allocated MCPWM Timer should be connected with a MCPWM operator by calling **mcpwm_operator_connect_timer()**, so that the operator can take that timer as its time base, and generate the required PWM waves. Make sure the MCPWM timer and operator are in the same group, otherwise, this function will return **ESP_ERR_INVALID_ARG** error.

**Comparator Operations and Events**

**Register Event Callbacks** The MCPWM comparator can inform the user when the timer counter equals to the compare value. If you have some function that should be called when this event happens, you should hook your function to the interrupt service routine by calling **mcpwm_comparator_register_event_callbacks()**. The callback function prototype is declared in **mcpwm_compare_event_cb_t**. All supported event callbacks are listed in the **mcpwm_comparator_event_callbacks_t**:

- **mcpwm_comparator_event_callbacks_t::on_reach** sets callback function for comparator when the timer counter equals to the compare value.

  The callback function will provide event specific data of type **mcpwm_compare_event_data_t** to the user. The callback function is called within the ISR context, so is should **not** attempt to block (e.g., make sure that only FreeRTOS APIs with **ISR** suffix is called within the function).

  The parameter **user_data** of **mcpwm_comparator_register_event_callbacks()** function is used to save user’s own context, it will be passed to the callback function directly.

  This function will lazy install interrupt service for the MCPWM comparator, whereas the service can only be removed in **mcpwm_del_comparator**.

**Set Compare Value** You can set the compare value for the MCPWM comparator at runtime by calling **mcpwm_comparator_set_compare_value()**. There’re a few points to note:
Chapter 2. API

- New compare value might won’t take effect immediately. The update time for the compare value is set by `mcpwm_comparator_config_t::update_cmp_on_tez` or `mcpwm_comparator_config_t::update_cmp_on_tep` or `mcpwm_comparator_config_t::update_cmp_on_sync`.
- Make sure the operator has connected to one MCPWM timer already by `mcpwm_operator_connect_timer()` otherwise, it will return error code `ESP_ERR_INVALID_STATE`.
- The compare value shouldn’t exceed timer’s count peak, otherwise, the compare event will never get triggered.

**Generator Actions on Events**

**Set Generator Action on Timer Event** One generator can set multiple actions on different timer events, by calling `mcpwm_generator_set_actions_on_timer_event()` with variable number of action configurations. The action configuration is defined in `mcpwm_gen_timer_event_action_t`:

- `mcpwm_gen_timer_event_action_t::direction` specifies the timer direction. The supported directions are listed in `mcpwm_timer_direction_t`.
- `mcpwm_gen_timer_event_action_t::event` specifies the timer event. The supported timer events are listed in `mcpwm_timer_event_t`.
- `mcpwm_gen_timer_event_action_t::action` specifies the generator action to be taken. The supported actions are listed in `mcpwm_generator_action_t`.

There’s a helper macro `MCPWM_GEN_TIMER_EVENT_ACTION` to simplify the construction of a timer event action entry.

Please note, the argument list of `mcpwm_generator_set_actions_on_timer_event()` must be terminated by `MCPWM_GEN_TIMER_EVENT_ACTION_END`.

**Set Generator Action on Compare Event** One generator can set multiple actions on different compare events, by calling `mcpwm_generator_set_actions_on_compare_event()` with variable number of action configurations. The action configuration is defined in `mcpwm_gen_compare_event_action_t`:

- `mcpwm_gen_compare_event_action_t::direction` specifies the timer direction. The supported directions are listed in `mcpwm_timer_direction_t`.
- `mcpwm_gen_compare_event_action_t::comparator` specifies the comparator handle. See `MCPWM Comparators` for how to allocate a comparator.
- `mcpwm_gen_compare_event_action_t::action` specifies the generator action to be taken. The supported actions are listed in `mcpwm_generator_action_t`.

There’s a helper macro `MCPWM_GEN_COMPARE_EVENT_ACTION` to simplify the construction of a compare event action entry.

Please note, the argument list of `mcpwm_generator_set_actions_on_compare_event()` must be terminated by `MCPWM_GEN_COMPARE_EVENT_ACTION_END`.

**Classical PWM Waveforms and Generator Configurations** This section will demonstrate the classical PWM waveforms that can be generated by the pair of the generators. The code snippet that is used to generate the waveforms is also provided below the diagram. Some general summary:

- The Symmetric or Asymmetric of the waveforms are determined by the count mode of the MCPWM timer.
- The active level of the waveform pair is determined by the level of the PWM with a smaller duty cycle.
- The period of the PWM waveform is determined by the timer’s period and count mode.
- The duty cycle of the PWM waveform is determined by the generator’s various action combinations.

### Asymmetric Single Edge Active High

![Diagram of Asymmetric Single Edge Active High PWM waveform]
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb, mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena,
        MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
        TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH),
        MCPWM_GEN_TIMER_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
        MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,
        MCPWM_GEN_ACTION_LOW),
        MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(genb,
        MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_
        TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH),
        MCPWM_GEN_TIMER_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb,
        MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,
        MCPWM_GEN_ACTION_LOW),
        MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}

Asymmetric Single Edge Active Low

static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb, mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
        MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,
        MCPWM_GEN_ACTION_HIGH),
        MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb,
        MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,
        MCPWM_GEN_ACTION_LOW),
        MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
Asymmetric Dual Edge Active Low

```c
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,
                            mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, 
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa, 
                    MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpa, 
                    MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb, 
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb, 
                    MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpb, 
                    MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
```

Symmetric Dual Edge Active Low

```c
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,
                            mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, 
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa, 
                    MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpa, 
                    MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb, 
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb, 
                    MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpb, 
                    MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
```
**Symmetric Dual Edge Complementary**

```c
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,
                             mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa,
                    MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpa,
                    MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb,
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb,
                    MCPWM_GEN_ACTION_LOW),
                    MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_DOWN, cmpb,
                    MCPWM_GEN_ACTION_HIGH),
                    MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
```

**Dead Time**  In power electronics, the rectifier and inverter are commonly used. This requires the use of rectifier bridge and inverter bridge. Each bridge arm has two power electronic devices, such as MOSFET, IGBT, etc. The two MOSFETs on the same arm can’t conduct at the same time, otherwise there will be a short circuit. The fact is that, although the PWM wave shows it is turning off the switch, but the MOSFET still needs a small time window to make that happen. This requires an extra delay to be added to the existing PWM wave that generated by setting `Generator Actions on Events`.

The dead-time driver works like a *decorator*, which is also reflected in the function parameters of `mcpwm_generator_set_dead_time()`, where it takes the primary generator handle (`in_generator`), and returns a generator (`out_generator`) after applying the dead-time. Please note, if the `out_generator` and `in_generator` are the same, it means we’re adding the time delay to the PWM waveform in a “in-place” fashion. In turn, if the `out_generator` and `in_generator` are different, it means we’re deriving a new PWM waveform from the existing `in_generator`.

Dead-time specific configuration is listed in the `mcpwm_dead_time_config_t` structure:

- `mcpwm_dead_time_config_t::posedge_delay_ticks` and `mcpwm_dead_time_config_t::negedge_delay_ticks` set the number of ticks to delay the PWM waveform on the rising and falling edge. Specifically, setting both of them to zero means to bypass the dead-time module. The resolution of the dead-time tick is the same to the timer that is connected with the operator by `mcpwm_operator_connect_timer()`.
- `mcpwm_dead_time_config_t::invert_output`: Whether to invert the signal after applying the dead-time, which can be used to control the delay edge polarity.

**备注：** It is also possible to generate the required dead time by setting `Generator Actions on Events`, especially by controlling edge placement using different comparators. However, if the more classical edge delay-based dead time with polarity control is required, then the dead-time submodule should be used.

**Classical PWM Waveforms and Dead Time Configurations**  This section will demonstrate the classical PWM waveforms that can be generated by the dead-time submodule. The code snippet that is used to generate the waveforms is also provided below the diagram.
Active High, Complementary

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb, 
                             mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, 
                   MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_ 
                   TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH), 
                   MCPWM_GEN_TIMER_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, 
                   MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa, 
                   MCPWM_GEN_ACTION_LOW), 
                   MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
```

Active High Complementary

```
static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 50,
        .negedge_delay_ticks = 0
    };
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    dead_time_config.posedge_delay_ticks = 0;
    dead_time_config.negedge_delay_ticks = 100;
    dead_time_config.flags.invert_output = true;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}
```

Active Low, Complementary

```
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb, 
                             mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, 
                   MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_ 
                   TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH), 
                   MCPWM_GEN_TIMER_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, 
                   MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa, 
                   MCPWM_GEN_ACTION_LOW), 
                   MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}
```

Active Low Complementary

```
static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
    ...
    }
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
}
```
```c

static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb, mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH), MCPWM_TIMER_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa, MCPWM_GEN_ACTION_LOW), MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = { .posedge_delay_ticks = 50, .negedge_delay_ticks = 0, };
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    dead_time_config.posedge_delay_ticks = 0;
    dead_time_config.negedge_delay_ticks = 100;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}
```

---

### Active High

![Active High Diagram](image)

### Active Low

![Active Low Diagram](image)
ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH), MCPWM_GEN_TIMER_EVENT_ACTION_END()));
ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa, MCPWM_GEN_ACTION_LOW), MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
  mcpwm_dead_time_config_t dead_time_config = {
    .posedge_delay_ticks = 50,
    .negedge_delay_ticks = 0,
    .flags.invert_output = true
  };
  ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
  dead_time_config.posedge_delay_ticks = 0;
  dead_time_config.negedge_delay_ticks = 100;
  ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}

static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb,
  mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
  ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH), MCPWM_GEN_TIMER_EVENT_ACTION_END()));
  ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpa, MCPWM_GEN_ACTION_LOW), MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
  ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(genb, MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH), MCPWM_GEN_TIMER_EVENT_ACTION_END()));
  ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb, MCPWM_GEN_ACTION_LOW), MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
  mcpwm_dead_time_config_t dead_time_config = {
    .posedge_delay_ticks = 50,
  };
}
Chapter 2. API参考

```c
static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb, mcpwm_cmpr_handle_t cmpa, mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH, MCPWM_GEN_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, cmpa, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_GEN_ACTION_LOW, MCPWM_GEN_ACTION_LOW, MCPWM_GEN_COMPARE_EVENT_ACTION_END())));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH, MCPWM_GEN_ACTION_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, cmpb, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, MCPWM_GEN_ACTION_LOW, MCPWM_GEN_ACTION_ACTION_END())));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 0,
        .negedge_delay_ticks = 0,
    };
    // generator_a bypass the deadtime module (no delay)
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    // apply dead time to generator_b
    dead_time_config.negedge_delay_ticks = 50;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}
```

FED on B, Bypass A

origin_A

origin_B

pwm_A

pwm_B

Falling Delay on PWMB, Bypass deadtime for PWMA
Rising and Falling Delay on PWMB, Bypass deadtime for PWMA

Bypass A, RED + FED on B

Origin_A

Origin_B

Pwm_A

Pwm_B

static void gen_action_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb, __mcpwm_cmpr_handle_t cmpa, __mcpwm_cmpr_handle_t cmpb)
{
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(gena, MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH, MCPWM_GEN_TIMER_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(gena, MCPWM_TIMER_DIRECTION_UP, cmpa, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP,(cmpb, MCPWM_GEN_ACTION_LOW), MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_timer_event(genb, MCPWM_TIMER_DIRECTION_UP, MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH, MCPWM_GEN_TIMER_EVENT_ACTION_END()));
    ESP_ERROR_CHECK(mcpwm_generator_set_actions_on_compare_event(genb, MCPWM_TIMER_DIRECTION_UP, cmpb, MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP, cmpb, MCPWM_GEN_ACTION_LOW), MCPWM_GEN_COMPARE_EVENT_ACTION_END()));
}

static void dead_time_config(mcpwm_gen_handle_t gena, mcpwm_gen_handle_t genb)
{
    mcpwm_dead_time_config_t dead_time_config = {
        .posedge_delay_ticks = 0,
        .negedge_delay_ticks = 0,
    };
    // generator_a bypass the deadtime module (no delay)
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, gena, &dead_time_config));
    // apply dead time on both edge for generator_b
    dead_time_config.negedge_delay_ticks = 50;
    dead_time_config.posedge_delay_ticks = 50;
    ESP_ERROR_CHECK(mcpwm_generator_set_dead_time(gena, genb, &dead_time_config));
}

Carrier Modulation The MCPWM operator has a carrier submodule that can be used if galvanic isolation from the motor driver is required (e.g. isolated digital power application) by passing the PWM output signals through transformers. Any of PWM output signals may be at 100% duty and not changing whenever motor is required to run steady at the full load. Coupling of non alternating signals with a transformer is problematic, so the signals are modulated by the carrier submodule to create an AC waveform, to make the coupling possible.

To configure the carrier submodule, you can call mcpwm_operator_apply_carrier(), and provide configuration structure mcpwm_carrier_config_t:

- mcpwm_carrier_config_t::frequency_hz: The carrier frequency in Hz.
- mcpwm_carrier_config_t::duty_cycle: The duty cycle of the carrier. Note that, the supported choices of duty cycle are discrete, the driver will search the nearest one based the user configuration.
- mcpwm_carrier_config_t::first_pulse_duration_us: The duration of the first pulse in microseconds. The resolution of the first pulse duration is determined by the carrier frequency you set in the
Chapter 2. API

mcpwm_carrier_config_t::frequency_hz. The first pulse duration can’t be zero, and it has to be at least one period of the carrier. A longer pulse width can help conduct the inductance quicker.

- mcpwm_carrier_config_t::invert_before_modulate and mcpwm_carrier_config_t::invert_after_modulate: Set whether to invert the carrier output before and after modulation.

Specifically, the carrier submodule can be disabled by calling mcpwm_operator_applyCarrier() with a NULL configuration.

Faults and Brake Actions  The MCPWM operator is able to sense external signals with information about failure of the motor, the power driver or any other device connected. These failure signals are encapsulated into MCPWM fault objects.

The user should determine possible failure modes of the motor and what action should be performed on detection of particular fault, e.g. drive all outputs low for a brushed motor, or lock current state for a stepper motor, etc. As result of this action the motor should be put into a safe state to reduce likelihood of a damage caused by the fault.

Set Operator Brake Mode on Fault  The way that MCPWM operator reacts to the fault is called Brake. The MCPWM operator can be configured to perform different brake modes for each fault object by calling mcpwm_operator_set_brake_on_fault(). Brake specific configuration is passed as a structure mcpwm_brake_config_t:

- mcpwm_brake_config_t::fault set which fault that the operator should react to.
- mcpwm_brake_config_t::brake_mode set the brake mode that should be used for the fault. The supported brake modes are listed in the mcpwm_operator_brake_mode_t. For MCPWM_OPER_BRAKE_MODE_CBC mode, the operator will recover itself automatically as long as the fault disappears. You can specify the recovery time in mcpwm_brake_config_t::cbc_recover_on_tez and mcpwm_brake_config_t::cbc_recover_on_tep. For MCPWM_OPER_BRAKE_MODE_OST mode, the operator can’t recover even though the fault disappears. User has to call mcpwm_operator_recover_from_fault() to manually recover it.

Set Generator Action on Brake Event  One generator can set multiple actions on different brake events, by calling mcpwm_generator_set_actions_on_brake_event() with variable number of action configurations. The action configuration is defined in mcpwm_gen_brake_event_action_t:

- mcpwm_gen_brake_event_action_t::direction specific the timer direction. The supported directions are listed in mcpwm_timer_direction_t.
- mcpwm_gen_brake_event_action_t::brake_mode specifies the brake mode. The supported brake modes are listed in the mcpwm_operator_brake_mode_t.
- mcpwm_gen_brake_event_action_t::action specifies the generator action to be taken. The supported actions are listed in mcpwm_generator_action_t.

There’s a helper macro MCPWM_GEN_BRAKE_EVENT_ACTION to simplify the construction of a brake event action entry.

Please note, the argument list of mcpwm_generator_set_actions_on_brake_event() must be terminated by MCPWM_GEN_BRAKE_EVENT_ACTION_END.

Register Fault Event Callbacks  The MCPWM fault detector can inform the user when it detects a valid fault or a fault signal disappears. If you have some function that should be called when such event happens, you should hook your function to the interrupt service routine by calling mcpwm_fault_register_event_callbacks(). The callback function prototype is declared in mcpwm_fault_event_cb_t. All supported event callbacks are listed in the mcpwm_fault_event_callbacks_t:

- mcpwm_fault_event_callbacks_t::on_fault_enter sets callback function that will be called when a fault is detected.
- mcpwm_fault_event_callbacks_t::on_fault_exit sets callback function that will be called when a fault is cleared.
Chapter 2. API

The callback function is called within the ISR context, so is should **not** attempt to block (e.g., make sure that only FreeRTOS APIs with `ISR` suffix is called within the function).

The parameter `user_data` of `mcpwm_fault_register_event_callbacks()` function is used to save user’s own context, it will be passed to the callback function directly.

This function will lazy install interrupt service for the MCPWM fault, whereas the service can only be removed in `mcpwm_del_fault`.

### Register Brake Event Callbacks

The MCPWM operator can inform the user when it going to take a brake action. If you have some function that should be called when this event happens, you should hook your function to the interrupt service routine by calling `mcpwm_operator_register_event_callbacks()`. The callback function prototype is declared in `mcpwm_brake_event_cb_t`. All supported event callbacks are listed in the `mcpwm_operator_event_callbacks_t`:

- `mcpwm_operator_event_callbacks_t::on_brake_cbc` sets callback function that will be called when the operator is going to take a CBC action.
- `mcpwm_operator_event_callbacks_t::on_brake_ost` sets callback function that will be called when the operator is going to take an OST action.

The callback function is called within the ISR context, so is should **not** attempt to block (e.g., make sure that only FreeRTOS APIs with `ISR` suffix is called within the function).

The parameter `user_data` of `mcpwm_operator_register_event_callbacks()` function is used to save user’s own context, it will be passed to the callback function directly.

This function will lazy install interrupt service for the MCPWM operator, whereas the service can only be removed in `mcpwm_del_operator`.

### Generator Force Actions

Software can override generator output level at runtime, by calling `mcpwm_generator_set_force_level()`. The software force level always has a higher priority than other event actions set in e.g. `mcpwm_generator_set_actions_on_timer_event()`.

- Set the `level` to -1 means to disable the force action, and the generator’s output level will be controlled by the event actions again.
- Set the `hold_on` to true, the force output level will keep alive, until it’s removed by assigning `level` to -1.
- Set the `hole_on` to false, the force output level will only be active for a short time, any upcoming event can override it.

### Synchronization

When a sync signal is taken by the MCPWM timer, the timer will be forced into a predefined phase, where the phase is determined by count value and count direction. You can set the sync phase by calling `mcpwm_timer_set_phase_on_sync()`. The sync phase configuration is defined in `mcpwm_timer_sync_phase_config_t` structure:

- `mcpwm_timer_sync_phase_config_t::sync_src` sets the sync signal source. See **MCPWM Sync Sources** for how to create a sync source object. Specifically, if this is set to NULL, the driver will disable the sync feature for the MCPWM timer.
- `mcpwm_timer_sync_phase_config_t::count_value` sets the count value to load when the sync signal is taken.
- `mcpwm_timer_sync_phase_config_t::direction` sets the count direction when the sync signal is taken.

Likewise, the MCPWM capture timer **MCPWM Capture Timer** can be synced as well. You can set the sync phase for the capture timer by calling `mcpwm_capture_timer_set_phase_on_sync()`. The sync phase configuration is defined in `mcpwm_capture_timer_sync_phase_config_t` structure:

- `mcpwm_capture_timer_sync_phase_config_t::sync_src` sets the sync signal source. See **MCPWM Sync Sources** for how to create a sync source object. Specifically, if this is set to NULL, the driver will disable the sync feature for the MCPWM capture timer.
- `mcpwm_capture_timer_sync_phase_config_t::count_value` sets the count value to load when the sync signal is taken.
**mcpwm_capture_timer_sync_phase_config_t::direction** sets the count direction when the sync signal is taken. Note that, different from MCPWM Timer, the capture timer can only support one count direction: **MCPWM_TIMER_DIRECTION_UP**.

```
static void example_setup_sync_strategy(mcpwm_timer_handle_t timers[])
{
    mcpwm_sync_handle_t gpio_sync_source = NULL;
    mcpwm_gpio_sync_src_config_t gpio_sync_config = {
        .group_id = 0,                                        // GPIO fault should be in the same group of...
        .gpio_num = EXAMPLE_SYNC_GPIO,
        .flags.pull_down = true,
        .flags.active_neg = false,                           // by default, a posedge pulse can trigger a...
        .sync_src = gpio_sync_source,
    };
    ESP_ERROR_CHECK(mcpwm_new_gpio_sync_src(&gpio_sync_config, &gpio_sync_source));

    mcpwm_timer_sync_phase_config_t sync_phase_config = {
        .count_value = 0,                                     // sync phase: target count value
        .direction = MCPWM_TIMER_DIRECTION_UP,                // sync phase: count direction
        .sync_src = gpio_sync_source,                         // sync source
    };
    for (int i = 0; i < 3; i++) {
        ESP_ERROR_CHECK(mcpwm_timer_set_phase_on_sync(timers[i], &sync_phase_config));
    }
}
```

**Capture** The basic functionality of MCPWM capture is to record the time when any pulse edge of the capture signal turns active. Then you can get the pulse width and convert it into other physical quantity like distance or speed.
in the capture callback function. For example, in the BLDC (Brushless DC, see figure below) scenario, we can use the capture submodule to sense the rotor position from Hall sensor.

![Diagram of MCPWM BLDC with Hall Sensor](image)

**Register Event Callbacks** The MCPWM capture channel can inform the user when there’s a valid edge detected on the signal. You have to register a callback function to get the timer count value of the capture moment, by calling `mcpwm_capture_channel_register_event_callbacks()`. The callback function prototype is declared in `mcpwm_capture_event_cb_t`. All supported capture callbacks are listed in the `mcpwm_capture_event_callbacks_t`:

- `mcpwm_capture_event_callbacks_t::on_cap` sets callback function for the capture channel when a valid edge is detected.

The callback function will provide event specific data of type `mcpwm_capture_event_data_t`, so that you can get the edge of the capture signal in `mcpwm_capture_event_data_t::cap_edge` and the count value of that moment in `mcpwm_capture_event_data_t::cap_value`. To convert the capture count into timestamp, you need to know the resolution of the capture timer by calling `mcpwm_capture_timer_get_resolution()`.

The callback function is called within the ISR context, so it should not attempt to block (e.g., make sure that only FreeRTOS APIs with `ISR` suffix is called within the function).

The parameter `user_data` of `mcpwm_capture_channel_register_event_callbacks()` function is used to save user’s own context, it will be passed to the callback function directly.

This function will lazy install interrupt service for the MCPWM capture channel, whereas the service can only be removed in `mcpwm_del_capture_channel`.

**Enable and Disable Capture Channel** The capture channel is not enabled after allocation by `mcpwm_new_capture_channel()`. You should call `mcpwm_capture_channel_enable()` and `mcpwm_capture_channel_disable()` accordingly to enable or disable the
channel. If the interrupt service is lazy installed during registering event callbacks for the channel in `mcpwm_capture_channel_register_event_callbacks()`, `mcpwm_capture_channel_enable()` will enable the interrupt service as well.

**Enable and Disable Capture Timer** Before doing IO control to the capture timer, user needs to enable the timer first, by calling `mcpwm_capture_timer_enable()`. Internally, this function will:

- switch the capture timer state from `init` to `enable`.
- acquire a proper power management lock if a specific clock source (e.g. APB clock) is selected. See also [Power management](#) for more information.

On the contrary, calling `mcpwm_capture_timer_disable()` will put the timer driver back to `init` state, and release the power management lock.

**Start and Stop Capture Timer** The basic IO operation of a capture timer is to start and stop. Calling `mcpwm_capture_timer_start()` can start the timer and calling `mcpwm_capture_timer_stop()` can stop the timer immediately.

**Trigger a Software Capture Event** Sometime, the software also wants to trigger a “fake” capture event. The `mcpwm_capture_channel_trigger_soft_catch()` is provided for that purpose. Please note that, even though it’s a “fake” capture event, it can still cause an interrupt, thus your capture event callback function will get invoked as well.

**Power Management** When power management is enabled (i.e. `CONFIG_PM_ENABLE` is on), the system will adjust the PLL, APB frequency before going into light sleep, thus potentially changing the period of a MCPWM timers’ counting step and leading to inaccurate time keeping.

However, the driver can prevent the system from changing APB frequency by acquiring a power management lock of type `ESP_PM_APB_FREQ_MAX`. Whenever the driver creates a MCPWM timer instance that has selected `MCPWM_TIMER_CLK_SRC_PLL160M` as its clock source, the driver will guarantee that the power management lock is acquired when enable the timer by `mcpwm_timer_enable()`. Likewise, the driver releases the lock when `mcpwm_timer_disable()` is called for that timer.

Likewise, Whenever the driver creates a MCPWM capture timer instance that has selected `MCPWM_CAPTURE_CLK_SRC_APB` as its clock source, the driver will guarantee that the power management lock is acquired when enable the timer by `mcpwm_capture_timer_enable()`. And will release the lock in `mcpwm_capture_timer_disable()`.

**IRAM Safe** By default, the MCPWM interrupt will be deferred when the Cache is disabled for reasons like writing/erasing Flash. Thus the event callback functions will not get executed in time, which is not expected in a real-time application.

There’s a Kconfig option `CONFIG_MCPWM_ISR_IRAM_SAFE` that will:

1. Enable the interrupt being serviced even when cache is disabled
2. Place all functions that used by the ISR into IRAM
3. Place driver object into DRAM (in case it’s mapped to PSRAM by accident)

This will allow the interrupt to run while the cache is disabled but will come at the cost of increased IRAM consumption.

There is another Kconfig option `CONFIG_MCPWM_CTRL_FUNC_IN_IRAM` that can put commonly used IO control functions into IRAM as well. So, these functions can also be executable when the cache is disabled. These IO control functions are as follows:

- `mcpwm_comparator_set_compare_value()`

---

2 Callback function and the sub-functions invoked by itself should also be placed in IRAM, users need to take care of this by themselves.
**Thread Safety**  The factory functions like `mcpwm_new_timer()` are guaranteed to be thread safe by the driver, which means, you can call it from different RTOS tasks without protection by extra locks.

The following functions are allowed to run under ISR context, as the driver uses a critical section to prevent them being called concurrently in the task and ISR.

- `mcpwm_comparator_set_compare_value()`

Other functions that are not related to Resource Allocation, are not thread safe. Thus, you should avoid calling them in different tasks without mutex protection.

**Kconfig Options**

- `CONFIG_MCPWM_ISR_IRAM_SAFE` controls whether the default ISR handler can work when cache is disabled, see IRAM Safe for more information.
- `CONFIG_MCPWM_CTRL_FUNC_IN_IRAM` controls where to place the MCPWM control functions (IRAM or flash), see IRAM Safe for more information.
- `CONFIG_MCPWM_ENABLE_DEBUG_LOG` is used to enabled the debug log output. Enable this option will increase the firmware binary size.

**Application Examples**

- Brushed DC motor speed control by PID algorithm: `peripherals/mcpwm/mcpwm_bdc_speed_control`
- BLDC motor control with hall sensor feedback: `peripherals/mcpwm/mcpwm_bldc_hall_control`
- Ultrasonic sensor (HC-SR04) distance measurement: `peripherals/mcpwm/mcpwm_capture_hc_sr04`
- Servo motor angle control: `peripherals/mcpwm/mcpwm_servo_control`
- MCPWM synchronization between timers: `peripherals/mcpwm/mcpwm_sync`

**API Reference**

**Header File**

- `components/driver/include/driver/mcpwm_timer.h`

**Functions**

`esp_err_t mcpwm_new_timer(const mcpwm_timer_config_t *config, mcpwm_timer_handle_t *ret_timer)`  
Create MCPWM timer.

参数  
- `config` - [in] MCPWM timer configuration  
- `ret_timer` - [out] Returned MCPWM timer handle

返回  
- ESP_OK: Create MCPWM timer successfully  
- ESP_ERR_INVALID_ARG: Create MCPWM timer failed because of invalid argument  
- ESP_ERR_NO_MEM: Create MCPWM timer failed because out of memory  
- ESP_ERR_NOT_FOUND: Create MCPWM timer failed because all hardware timers are used up and no more free one  
- ESP_FAIL: Create MCPWM timer failed because of other error

`esp_err_t mcpwm_del_timer(mcpwm_timer_handle_t timer)`  
Delete MCPWM timer.

参数  
- `timer` - [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`

返回  
- ESP_OK: Delete MCPWM timer successfully  
- ESP_ERR_INVALID_ARG: Delete MCPWM timer failed because of invalid argument  
- ESP_ERR_INVALID_STATE: Delete MCPWM timer failed because timer is not in init state  
- ESP_FAIL: Delete MCPWM timer failed because of other error
**esp_err_t mcpwm_timer_enable** *(mcpwm_timer_handle_t timer)*

Enable MCPWM timer.

- **timer** - [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`
- **返回**
  - ESP_OK: Enable MCPWM timer successfully
  - ESP_ERR_INVALID_ARG: Enable MCPWM timer failed because of invalid argument
  - ESP_ERR_INVALID_STATE: Enable MCPWM timer failed because timer is enabled already
  - ESP_FAIL: Enable MCPWM timer failed because of other error

**esp_err_t mcpwm_timer_disable** *(mcpwm_timer_handle_t timer)*

Disable MCPWM timer.

- **timer** - [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`
- **返回**
  - ESP_OK: Disable MCPWM timer successfully
  - ESP_ERR_INVALID_ARG: Disable MCPWM timer failed because of invalid argument
  - ESP_ERR_INVALID_STATE: Disable MCPWM timer failed because timer is disabled already
  - ESP_FAIL: Disable MCPWM timer failed because of other error

**esp_err_t mcpwm_timer_start_stop** *(mcpwm_timer_handle_t timer, mcpwm_timer_start_stop_cmd_t command)*

Send specific start/stop commands to MCPWM timer.

- **timer** - [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`
- **command** - [in] Supported command list for MCPWM timer
- **返回**
  - ESP_OK: Start or stop MCPWM timer successfully
  - ESP_ERR_INVALID_ARG: Start or stop MCPWM timer failed because of invalid argument
  - ESP_ERR_INVALID_STATE: Start or stop MCPWM timer failed because timer is not enabled
  - ESP_FAIL: Start or stop MCPWM timer failed because of other error

**esp_err_t mcpwm_timer_register_event_callbacks** *(mcpwm_timer_handle_t timer, const mcpwm_timer_event_callbacks_t *cbs, void *user_data)*

Set event callbacks for MCPWM timer.

**备注:** The first call to this function needs to be before the call to `mcpwm_timer_enable`

**备注:** User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

- **参数**
  - **timer** - [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`
  - **cbs** - [in] Group of callback functions
  - **user_data** - [in] User data, which will be passed to callback functions directly
- **返回**
  - ESP_OK: Set event callbacks successfully
  - ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
  - ESP_ERR_INVALID_STATE: Set event callbacks failed because timer is not in init state
  - ESP_FAIL: Set event callbacks failed because of other error
**esp_err_t mcpwm_timer_set_phase_on_sync**

```c
mcpwm_timer_set_phase_on_sync(mcpwm_timer_handle_t timer, const mcpwm_timer_sync_phase_config_t *config)
```

Sets sync phase for MCPWM timer.

- **timer** - [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`
- **config** - [in] MCPWM timer sync phase configuration

**Return**
- ESP_OK: Set sync phase for MCPWM timer successfully
- ESP_ERR_INVALID_ARG: Set sync phase for MCPWM timer failed because of invalid argument
- ESP_FAIL: Set sync phase for MCPWM timer failed because of other error

**Structures**

- **struct mcpwm_timer_event_callbacks_t**
  - Group of supported MCPWM timer event callbacks.

  备注：The callbacks are all running under ISR environment

**Public Members**

- **mcpwm_timer_event_cb_t on_full**
  - callback function when MCPWM timer counts to peak value

- **mcpwm_timer_event_cb_t on_empty**
  - callback function when MCPWM timer counts to zero

- **mcpwm_timer_event_cb_t on_stop**
  - callback function when MCPWM timer stops

- **struct mcpwm_timer_config_t**
  - MCPWM timer configuration.

  **Public Members**

- **int group_id**
  - Specify from which group to allocate the MCPWM timer

- **mcpwm_timer_clock_source_t clk_src**
  - MCPWM timer clock source

- **uint32_t resolution_hz**
  - Counter resolution in Hz, ranges from around 300KHz to 80MHz. The step size of each count tick equals to \((1 / \text{resolution}_hz)\) seconds

- **mcpwm_timer_count_mode_t count_mode**
  - Count mode
uint32_t **period_ticks**
Number of count ticks within a period

uint32_t **update_period_on_empty**
Whether to update period when timer counts to zero

uint32_t **update_period_on_sync**
Whether to update period on sync event

struct **mcpwm_timer_config_t**::[anonymous] **flags**
Extra configuration flags for timer

struct **mcpwm_timer_sync_phase_config_t**
MCPWM Timer sync phase configuration.

**Public Members**

**mcpwm_sync_handle_t** **sync_src**
The sync event source. Set to NULL will disable the timer being synced by others

uint32_t **count_value**
The count value that should lock to upon sync event

**mcpwm_timer_direction_t** **direction**
The count direction that should lock to upon sync event

**Header File**

- components/driver/include/driver/mcpwm_oper.h

**Functions**

```c
esp_err_t mcpwm_new_operator (const mcpwm_operator_config_t *config, mcpwm_oper_handle_t *ret_oper)
```
Create MCPWM operator.

参数
- **config** - [in] MCPWM operator configuration
- **ret_oper** - [out] Returned MCPWM operator handle

返回
- ESP_OK: Create MCPWM operator successfully
- ESP_ERR_INVALID_ARG: Create MCPWM operator failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM operator failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM operator failed because can’t find free resource
- ESP_FAIL: Create MCPWM operator failed because of other error

```c
esp_err_t mcpwm_del_operator (mcpwm_oper_handle_t oper)
```
Delete MCPWM operator.

参数 **oper** - [in] MCPWM operator, allocated by mcpwm_new_operator()

返回
- ESP_OK: Delete MCPWM operator successfully
Chapter 2. API 参考

- ESP_ERR_INVALID_ARG: Delete MCPWM operator failed because of invalid argument
- ESP_FAIL: Delete MCPWM operator failed because of other error

*esp_err_t* `mcpwm_operator_connect_timer`(mcpwm_oper_handle_t `oper`, mcpwm_timer_handle_t `timer`)

Connect MCPWM operator and timer, so that the operator can be driven by the timer.

参数

- `oper` : [in] MCPWM operator handle, allocated by `mcpwm_new_operator()`
- `timer` : [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`

返回

- ESP_OK: Connect MCPWM operator and timer successfully
- ESP_ERR_INVALID_ARG: Connect MCPWM operator and timer failed because of invalid argument
- ESP_FAIL: Connect MCPWM operator and timer failed because of other error

*esp_err_t* `mcpwm_operator_set_brake_on_fault`(mcpwm_oper_handle_t `oper`, const mcpwm_brake_config_t *`config`)

Set brake method for MCPWM operator.

参数

- `oper` : [in] MCPWM operator, allocated by `mcpwm_new_operator()`
- `config` : [in] MCPWM brake configuration

返回

- ESP_OK: Set trip for operator successfully
- ESP_ERR_INVALID_ARG: Set trip for operator failed because of invalid argument
- ESP_FAIL: Set trip for operator failed because of other error

*esp_err_t* `mcpwm_operator_recover_from_fault`(mcpwm_oper_handle_t `oper`, mcpwm_fault_handle_t `fault`)

Try to make the operator recover from fault.

备注: To recover from fault or escape from trip, you make sure the fault signal has dissappeared already. Otherwise the recovery can’t succeed.

参数

- `oper` : [in] MCPWM operator handle, allocated by `mcpwm_new_operator()`
- `fault` : [in] MCPWM fault handle

返回

- ESP_OK: Recover from fault successfully
- ESP_ERR_INVALID_ARG: Recover from fault failed because of invalid argument
- ESP_ERR_INVALID_STATE: Recover from fault failed because the fault source is still active
- ESP_FAIL: Recover from fault failed because of other error

*esp_err_t* `mcpwm_operator_register_event_callbacks`(mcpwm_oper_handle_t `oper`, const mcpwm_operator_event_callbacks_t *`cbs`, void *`user_data`)

Set event callbacks for MCPWM operator.

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.
Chapter 2. API

- **user_data** - [in] User data, which will be passed to callback functions directly

返回
- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

```c
esp_err_t mcpwm_operator_apply_carrier (mcpwm_oper_handle_t oper, const mcpwm_carrier_config_t *config)
```

Apply carrier feature for MCPWM operator.

参数
- oper - [in] MCPWM operator, allocated by `mcpwm_new_operator()`
- config - [in] MCPWM carrier specific configuration

返回
- ESP_OK: Set carrier for operator successfully
- ESP_ERR_INVALID_ARG: Set carrier for operator failed because of invalid argument
- ESP_FAIL: Set carrier for operator failed because of other error

**Structures**

```c
struct mcpwm_operator_config_t
```

MCPWM operator configuration.

**Public Members**

```c
int group_id
```

Specify from which group to allocate the MCPWM operator

```c
uint32_t update_gen_action_on_tez
```

Whether to update generator action when timer counts to zero

```c
uint32_t update_gen_action_on_tep
```

Whether to update generator action when timer counts to peak

```c
uint32_t update_gen_action_on_sync
```

Whether to update generator action on sync event

```c
uint32_t update_dead_time_on_tez
```

Whether to update dead time when timer counts to zero

```c
uint32_t update_dead_time_on_tep
```

Whether to update dead time when timer counts to peak

```c
uint32_t update_dead_time_on_sync
```

Whether to update dead time on sync event

```c
struct mcpwm_operator_config_t::[anonymous] flags
```

Extra configuration flags for operator

```c
struct mcpwm_brake_config_t
```

MCPWM brake configuration structure.
Public Members

`mcpwm_fault_handle_t fault`  
Which fault causes the operator to brake

`mcpwm_operator_brake_mode_t brake_mode`  
Brake mode

`uint32_t cbc_recover_on_tez`  
Recovery CBC brake state on tez event

`uint32_t cbc_recover_on_tep`  
Recovery CBC brake state on tep event

`struct mcpwm_brake_config_t::[anonymous] flags`  
Extra flags for brake configuration

`struct mcpwm_operator_event_callbacks_t`  
Group of supported MCPWM operator event callbacks.

---

备注: The callbacks are all running under ISR environment

Public Members

`mcpwm_brake_event_cb_t on_brake_cbc`  
callback function when mcpwm operator brakes in CBC

`mcpwm_brake_event_cb_t on_brake_ost`  
callback function when mcpwm operator brakes in OST

`struct mcpwm_carrier_config_t`  
MCPWM carrier configuration structure.

Public Members

`uint32_t frequency_hz`  
Carrier frequency in Hz

`uint32_t first_pulse_duration_us`  
The duration of the first PWM pulse, in us

`float duty_cycle`  
Carrier duty cycle

`uint32_t invert_before_modulate`  
Invert the raw signal
uint32_t invert_after_modulate
Invert the modulated signal

struct mcpwm_carrier_config_t::{anonymous} flags
Extra flags for carrier configuration

Header File
- components/driver/include/driver/mcpwm_cmpr.h

Functions

esp_err_t mcpwm_new_comparator (mcpwm_oper_handle_t oper, const mcpwm_comparator_config_t *config, mcpwm_cmpr_handle_t *ret_cmpr)
Create MCPWM comparator.
参数
- oper - [in] MCPWM operator, allocated by mcpwm_new_operator(), the new comparator will be allocated from this operator
- config - [in] MCPWM comparator configuration
- ret_cmpr - [out] Returned MCPWM comparator
返回
- ESP_OK: Create MCPWM comparator successfully
- ESP_ERR_INVALID_ARG: Create MCPWM comparator failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM comparator failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM comparator failed because can’t find free resource
- ESP_FAIL: Create MCPWM comparator failed because of other error

esp_err_t mcpwm_del_comparator (mcpwm_cmpr_handle_t cmpr)
Delete MCPWM comparator.
参数 cmpr - [in] MCPWM comparator handle, allocated by mcpwm_new_comparator()
返回
- ESP_OK: Delete MCPWM comparator successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM comparator failed because of invalid argument
- ESP_FAIL: Delete MCPWM comparator failed because of other error

esp_err_t mcpwm_comparator_register_event_callbacks (mcpwm_cmpr_handle_t cmpr, const mcpwm_comparator_event_callbacks_t *cbs, void *user_data)
Set event callbacks for MCPWM comparator.

备注: User can deregister a previously registered callback by calling this function and setting the callback member in the cbs structure to NULL.

参数
- cmpr - [in] MCPWM comparator handle, allocated by mcpwm_new_comparator()
- cbs - [in] Group of callback functions
- user_data - [in] User data, which will be passed to callback functions directly
返回
- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error
esp_err_t mcpwm_comparator_set_compare_value(mcpwm_cmpr_handle_t cmpr, uint32_t cmp_ticks)

Set MCPWM comparator’s compare value.

参数
- cmpr - [in] MCPWM comparator handle, allocated by mcpwm_new_comparator()
- cmp_ticks - [in] The new compare value

返回
- ESP_OK: Set MCPWM compare value successfully
- ESP_ERR_INVALID_ARG: Set MCPWM compare value failed because of invalid argument (e.g. the cmp_ticks is out of range)
- ESP_ERR_INVALID_STATE: Set MCPWM compare value failed because the operator doesn’t have a timer connected
- ESP_FAIL: Set MCPWM compare value failed because of other error

Structures

struct mcpwm_comparator_config_t

MCPWM comparator configuration.

Public Members

uint32_t update_cmp_on_tez

Whether to update compare value when timer count equals to zero (tez)

uint32_t update_cmp_on_tep

Whether to update compare value when timer count equals to peak (tep)

uint32_t update_cmp_on_sync

Whether to update compare value on sync event

struct mcpwm_comparator_config_t::[anonymous] flags

Extra configuration flags for comparator

struct mcpwm_comparator_event_callbacks_t

Group of supported MCPWM compare event callbacks.

备注: The callbacks are all running under ISR environment

Public Members

mcpwm_compare_event_cb_t on_reach

ISR callback function which would be invoked when counter reaches compare value

Header File

- components/driver/include/driver/mcpwm_gen.h
Functions

\[ \text{esp_err_t mcpwm_new_generator} \left( \text{mcpwm_oper_handle_t} \ oper, \ \text{const mcpwm_generator_config_t} \ *\text{config}, \ 
\text{mcpwm_gen_handle_t} \ *\text{ret_gen} \right) \]

Allocate MCPWM generator from given operator.

参数
- \text{oper} - [in] MCPWM operator, allocated by \text{mcpwm_new_operator}()
- \text{config} - [in] MCPWM generator configuration
- \text{ret_gen} - [out] Returned MCPWM generator

返回
- ESP_OK: Create MCPWM generator successfully
- ESP_ERR_INVALID_ARG: Create MCPWM generator failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM generator failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM generator failed because can’t find free resource
- ESP_FAIL: Create MCPWM generator failed because of other error

\[ \text{esp_err_t mcpwm_del_generator} \left( \text{mcpwm_gen_handle_t} \ gen \right) \]

Delete MCPWM generator.

参数 \text{gen} - [in] MCPWM generator handle, allocated by \text{mcpwm_new_generator}()

返回
- ESP_OK: Delete MCPWM generator successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM generator failed because of invalid argument
- ESP_FAIL: Delete MCPWM generator failed because of other error

\[ \text{esp_err_t mcpwm_generator_set_force_level} \left( \text{mcpwm_gen_handle_t} \ gen, \ \text{int} \ level, \ \text{bool} \ \text{hold_on} \right) \]

Set force level for MCPWM generator.

备注: The force level will be applied to the generator immediately, regardless any other events that would change the generator’s behaviour.

备注: If the \text{hold_on} is true, the force level will retain forever, until user removes the force level by setting the force level to -1.

备注: If the \text{hold_on} is false, the force level can be overridden by the next event action.

参数
- \text{gen} - [in] MCPWM generator handle, allocated by \text{mcpwm_new_generator}()
- \text{level} - [in] GPIO level to be applied to MCPWM generator, specially, -1 means to remove the force level
- \text{hold_on} - [in] Whether the forced PWM level should retain (i.e. will remain unchanged until manually remove the force level)

返回
- ESP_OK: Set force level for MCPWM generator successfully
- ESP_ERR_INVALID_ARG: Set force level for MCPWM generator failed because of invalid argument
- ESP_FAIL: Set force level for MCPWM generator failed because of other error

\[ \text{esp_err_t mcpwm_generator_set_actions_on_timer_event} \left( \text{mcpwm_gen_handle_t} \ gen, \ 
\text{mcpwm_gen_timer_event_action_t} \ ev_act, \ldots \right) \]

Set generator actions on different MCPWM timer events.
参数
• gen [in] MCPWM generator handle, allocated by mcpwm_new_generator()
• ev_act [in] MCPWM timer event action list, must be terminated by MCPWM_GEN TIMER_EVENT_ACTION_END()

返回
• ESP_OK: Set generator actions successfully
• ESP_ERR_INVALID_ARG: Set generator actions failed because of invalid argument
• ESP_ERR_INVALID_STATE: Set generator actions failed because of timer is not connected to operator
• ESP_FAIL: Set generator actions failed because of other error

esp_err_t mcpwm_generator_set_actions_on_compare_event (mcpwm_gen_handle_t generator,
                      mcpwm_gen_compare_event_action_t ev_act, ...)

Set generator actions on different MCPWM compare events.

参数
• generator [in] MCPWM generator handle, allocated by mcpwm_new_generator()
• ev_act [in] MCPWM compare event action list, must be terminated by MCPWM GEN_COMPARE_EVENT_ACTION_END()

返回
• ESP_OK: Set generator actions successfully
• ESP_ERR_INVALID_ARG: Set generator actions failed because of invalid argument
• ESP_FAIL: Set generator actions failed because of other error

esp_err_t mcpwm_generator_set_actions_on_brake_event (mcpwm_gen_handle_t generator,
                                      mcpwm_gen_brake_event_action_t ev_act, ...)

Set generator actions on different MCPWM brake events.

参数
• generator [in] MCPWM generator handle, allocated by mcpwm_new_generator()
• ev_act [in] MCPWM brake event action list, must be terminated by MCPWM_GEN_BRAKE_EVENT_ACTION_END()

返回
• ESP_OK: Set generator actions successfully
• ESP_ERR_INVALID_ARG: Set generator actions failed because of invalid argument
• ESP_FAIL: Set generator actions failed because of other error

esp_err_t mcpwm_generator_set_dead_time (mcpwm_gen_handle_t in_generator, mcpwm_gen_handle_t out_generator,
                                       const mcpwm_dead_time_config_t *config)

Set dead time for MCPWM generator.

参数
• in_generator [in] MCPWM generator, before adding the dead time
• out_generator [in] MCPWM generator, after adding the dead time
• config [in] MCPWM dead time configuration

返回
• ESP_OK: Set dead time for MCPWM generator successfully
• ESP_ERR_INVALID_ARG: Set dead time for MCPWM generator failed because of invalid argument
• ESP_FAIL: Set dead time for MCPWM generator failed because of other error

Structures

struct mcpwm_generator_config_t
MCPWM generator configuration.
Public Members

int gen_gpio_num
    The GPIO number used to output the PWM signal

uint32_t invert_pwm
    Whether to invert the PWM signal (done by GPIO matrix)

uint32_t io_loop_back
    For debug/test, the signal output from the GPIO will be fed to the input path as well

struct mcpwm_generator_config_t::[anonymous] flags
    Extra configuration flags for generator

Public Members

mcpwm_timer_direction_t direction
    Timer direction

mcpwm_timer_event_t event
    Timer event

mcpwm_generator_action_t action
    Generator action should perform

struct mcpwm_gen_timer_event_action_t
    Generator action on specific timer event.

Public Members

mcpwm_timer_direction_t direction
    Timer direction

mcpwm_cmpr_handle_t comparator
    Comparator handle

mcpwm_generator_action_t action
    Generator action should perform

struct mcpwm_gen_compare_event_action_t
    Generator action on specific comparator event.

Public Members

mcpwm_timer_direction_t direction
    Timer direction

mcpwm_gen_brake_event_action_t
    Generator action on specific brake event.
Public Members

**mcpwm_timer_direction_t** `direction`
Timer direction

**mcpwm_operator_brake_mode_t** `brake_mode`
Brake mode

**mcpwm_generator_action_t** `action`
Generator action should perform

```c
struct mcpwm_dead_time_config_t
MCPWM dead time configuration structure.
```

Public Members

```c
uint32_t `posedge_delay_ticks`
delay time applied to rising edge, 0 means no rising delay time

uint32_t `negedge_delay_ticks`
delay time applied to falling edge, 0 means no falling delay time

uint32_t `invert_output`
Invert the signal after applied the dead time

struct mcpwm_dead_time_config_t::[anonymous] flags
Extra flags for dead time configuration
```

Macros

```c
MCPWM_GEN_TIMER_EVENT_ACTION (dir, ev, act)
Help macros to construct a mcpwm_gen_timer_event_action_t entry.

MCPWM_GEN_TIMER_EVENT_ACTION_END ()

MCPWM_GEN_COMPARE_EVENT_ACTION (dir, cmp, act)
Help macros to construct a mcpwm_gen_compare_event_action_t entry.

MCPWM_GEN_COMPARE_EVENT_ACTION_END ()

MCPWM_GEN_BRAKE_EVENT_ACTION (dir, mode, act)
Help macros to construct a mcpwm_gen_brake_event_action_t entry.

MCPWM_GEN_BRAKE_EVENT_ACTION_END ()
```

Header File

- `components/driver/include/driver/mcpwm_fault.h`
Chapter 2. API 参考

Functions

```c
esp_err_t mcpwm_new_gpio_fault (const mcpwm_gpio_fault_config_t *config, mcpwm_fault_handle_t *ret_fault)
```

Create MCPWM GPIO fault.

参数

- `config` [in] MCPWM GPIO fault configuration
- `ret_fault` [out] Returned GPIO fault handle

返回

- ESP_OK: Create MCPWM GPIO fault successfully
- ESP_ERR_INVALID_ARG: Create MCPWM GPIO fault failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM GPIO fault failed because out of memory
- ESP_FAIL: Create MCPWM GPIO fault failed because of other error

```c
esp_err_t mcpwm_new_soft_fault (const mcpwm_soft_fault_config_t *config, mcpwm_fault_handle_t *ret_fault)
```

Create MCPWM software fault.

参数

- `config` [in] MCPWM software fault configuration
- `ret_fault` [out] Returned software fault handle

返回

- ESP_OK: Create MCPWM software fault successfully
- ESP_ERR_INVALID_ARG: Create MCPWM software fault failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM software fault failed because out of memory
- ESP_FAIL: Create MCPWM software fault failed because of other error

```c
esp_err_t mcpwm_del_fault (mcpwm_fault_handle_t fault)
```

Delete MCPWM fault.

参数 `fault` [in] MCPWM fault handle allocated by `mcpwm_new_gpio_fault()` or `mcpwm_new_soft_fault()`

返回

- ESP_OK: Delete MCPWM fault successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM fault failed because of invalid argument
- ESP_FAIL: Delete MCPWM fault failed because of other error

```c
esp_err_t mcpwm_soft_fault_activate (mcpwm_fault_handle_t fault)
```

Activate the software fault, trigger the fault event for once.

参数 `fault` [in] MCPWM soft fault, allocated by `mcpwm_new_soft_fault()`

返回

- ESP_OK: Activate MCPWM software fault successfully
- ESP_ERR_INVALID_ARG: Activate MCPWM software fault event failed because of invalid argument
- ESP_FAIL: Activate MCPWM software fault event failed because of other error

```c
esp_err_t mcpwm_fault_register_event_callbacks (mcpwm_fault_handle_t fault, const mcpwm_fault_event_callbacks_t *cbs, void *user_data)
```

Set event callbacks for MCPWM fault.

备注：User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.
Chapter 2. API

- **fault** - [in] MCPWM GPIO fault handle, allocated by `mcpwm_new_gpio_fault()`
- **cbs** - [in] Group of callback functions
- **user_data** - [in] User data, which will be passed to callback functions directly

### Structures

**struct mcpwm_gpio_fault_config_t**

MCPWM GPIO fault configuration structure.

#### Public Members

- **int group_id**
  - In which MCPWM group that the GPIO fault belongs to

- **int gpio_num**
  - GPIO used by the fault signal

- **uint32_t active_level**
  - On which level the fault signal is treated as active

- **uint32_t io_loop_back**
  - For debug/test, the signal output from the GPIO will be fed to the input path as well

- **uint32_t pull_up**
  - Whether to pull up internally

- **uint32_t pull_down**
  - Whether to pull down internally

- **struct mcpwm_gpio_fault_config_t::[anonymous] flags**
  - Extra configuration flags for GPIO fault

**struct mcpwm_soft_fault_config_t**

MCPWM software fault configuration structure.

**struct mcpwm_fault_event_callbacks_t**

Group of supported MCPWM fault event callbacks.

### 注释:
The callbacks are all running under ISR environment

#### Public Members
Chapter 2. API 参考

`mcpwm_fault_event_cb_t on_fault_enter`
ISR callback function that would be invoked when fault signal becomes active

`mcpwm_fault_event_cb_t on_fault_exit`
ISR callback function that would be invoked when fault signal becomes inactive

**Header File**
- components/driver/include/driver/mcpwm_sync.h

**Functions**

```c
esp_err_t mcpwm_new_timer_sync_src(mcpwm_timer_handle_t timer, const mcpwm_timer_sync_src_config_t *config, mcpwm_sync_handle_t *ret_sync)
```
Create MCPWM timer sync source.

**参数**
- `timer` [in] MCPWM timer handle, allocated by `mcpwm_new_timer()`
- `config` [in] MCPWM timer sync source configuration
- `ret_sync` [out] Returned MCPWM sync handle

**返回**
- ESP_OK: Create MCPWM timer sync source successfully
- ESP_ERR_INVALID_ARG: Create MCPWM timer sync source failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM timer sync source failed because out of memory
- ESP_ERR_INVALID_STATE: Create MCPWM timer sync source failed because the timer has created a sync source before
- ESP_FAIL: Create MCPWM timer sync source failed because of other error

```c
esp_err_t mcpwm_new_gpio_sync_src(const mcpwm_gpio_sync_src_config_t *config, mcpwm_sync_handle_t *ret_sync)
```
Create MCPWM GPIO sync source.

**参数**
- `config` [in] MCPWM GPIO sync source configuration
- `ret_sync` [out] Returned MCPWM GPIO sync handle

**返回**
- ESP_OK: Create MCPWM GPIO sync source successfully
- ESP_ERR_INVALID_ARG: Create MCPWM GPIO sync source failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM GPIO sync source failed because out of memory
- ESP_ERR_NOT_FOUND: Create MCPWM GPIO sync source failed because can’t find free resource
- ESP_FAIL: Create MCPWM GPIO sync source failed because of other error

```c
esp_err_t mcpwm_new_soft_sync_src(const mcpwm_soft_sync_config_t *config, mcpwm_sync_handle_t *ret_sync)
```
Create MCPWM software sync source.

**参数**
- `config` [in] MCPWM software sync source configuration
- `ret_sync` [out] Returned software sync handle

**返回**
- ESP_OK: Create MCPWM software sync source successfully
- ESP_ERR_INVALID_ARG: Create MCPWM software sync failed because of invalid argument
- ESP_ERR_NO_MEM: Create MCPWM software sync failed because out of memory
- ESP_FAIL: Create MCPWM software sync failed because of other error

Submit Document Feedback
Release v5.1-dev-2066-g7869f4e151
esp_err_t mcpwm_del_sync_src (mcpwm_sync_handle_t sync)
Delete MCPWM sync source.

参数 sync - [in] MCPWM sync handle, allocated by mcpwm_new_timer_sync_src() or mcpwm_new_gpio_sync_src() or mcpwm_new_soft_sync_src()

返回
- ESP_OK: Delete MCPWM sync source successfully
- ESP_ERR_INVALID_ARG: Delete MCPWM sync source failed because of invalid argument
- ESP_FAIL: Delete MCPWM sync source failed because of other error

esp_err_t mcpwm_soft_sync_activate (mcpwm_sync_handle_t sync)
Activate the software sync, trigger the sync event for once.

参数 sync - [in] MCPWM soft sync handle, allocated by mcpwm_new_soft_sync_src()

返回
- ESP_OK: Trigger MCPWM software sync event successfully
- ESP_ERR_INVALID_ARG: Trigger MCPWM software sync event failed because of invalid argument
- ESP_FAIL: Trigger MCPWM software sync event failed because of other error

Structures

struct mcpwm_timer_sync_src_config_t
MCPWM timer sync source configuration.

Public Members

mcpwm_timer_event_t timer_event
Timer event, upon which MCPWM timer will generate the sync signal

uint32_t propagate_input_sync
The input sync signal would be routed to its sync output

struct mcpwm_timer_sync_src_config_t::[anonymous] flags
Extra configuration flags for timer sync source

struct mcpwm_gpio_sync_src_config_t
MCPWM GPIO sync source configuration.

Public Members

int group_id
MCPWM group ID

int gpio_num
GPIO used by sync source

uint32_t active_neg
Whether the sync signal is active on negedge, by default, the sync signal’s posedge is treated as active
uint32_t io_loop_back
    For debug/test, the signal output from the GPIO will be fed to the input path as well

uint32_t pull_up
    Whether to pull up internally

uint32_t pull_down
    Whether to pull down internally

struct mcpwm_gpio_sync_src_config_t::[anonymous] flags
    Extra configuration flags for GPIO sync source

struct mcpwm_soft_sync_config_t
    MCPWM software sync configuration structure.

Header File
• components/driver/include/driver/mcpwm_cap.h

Functions

esp_err_t mcpwm_new_capture_timer (const mcpwm_capture_timer_config_t *config,
                                    mcpwm_cap_timer_handle_t *ret_cap_timer)

Create MCPWM capture timer.

参数
• config [in] MCPWM capture timer configuration
• ret_cap_timer [out] Returned MCPWM capture timer handle

返回
• ESP_OK: Create MCPWM capture timer successfully
• ESP_ERR_INVALID_ARG: Create MCPWM capture timer failed because of invalid argument
• ESP_ERR_NO_MEM: Create MCPWM capture timer failed because out of memory
• ESP_ERR_NOT_FOUND: Create MCPWM capture timer failed because can’t find free resource
• ESP_FAIL: Create MCPWM capture timer failed because of other error

esp_err_t mcpwm_del_capture_timer (mcpwm_cap_timer_handle_t cap_timer)

Delete MCPWM capture timer.

参数 cap_timer [in] MCPWM capture timer, allocated by mcpwm_new_capture_timer()

返回
• ESP_OK: Delete MCPWM capture timer successfully
• ESP_ERR_INVALID_ARG: Delete MCPWM capture timer failed because of invalid argument
• ESP_FAIL: Delete MCPWM capture timer failed because of other error

esp_err_t mcpwm_capture_timer_enable (mcpwm_cap_timer_handle_t cap_timer)

Enable MCPWM capture timer.

参数 cap_timer [in] MCPWM capture timer handle, allocated by mcpwm_new_capture_timer()

返回
• ESP_OK: Enable MCPWM capture timer successfully
• ESP_ERR_INVALID_ARG: Enable MCPWM capture timer failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable MCPWM capture timer failed because timer is enabled already
- ESP_FAIL: Enable MCPWM capture timer failed because of other error

```c
esp_err_t mcpwm_capture_timer_disable (mcpwm_cap_timer_handle_t cap_timer)
```

Disable MCPWM capture timer.

<table>
<thead>
<tr>
<th>参数</th>
<th>cap_timer</th>
<th>- [in]</th>
<th>MCPWM capture timer handle, allocated by mcpwm_new_capture_timer()</th>
</tr>
</thead>
<tbody>
<tr>
<td>返回</td>
<td>ESP_OK: Disable MCPWM capture timer successfully</td>
<td></td>
<td>ESP_ERR_INVALID_ARG: Disable MCPWM capture timer failed because of invalid argument</td>
</tr>
<tr>
<td></td>
<td>ESP_FAIL: Disable MCPWM capture timer failed because timer is disabled already</td>
<td></td>
<td>ESP_FAIL: Disable MCPWM capture timer failed because of other error</td>
</tr>
</tbody>
</table>

```c
esp_err_t mcpwm_capture_timer_start (mcpwm_cap_timer_handle_t cap_timer)
```

Start MCPWM capture timer.

<table>
<thead>
<tr>
<th>参数</th>
<th>cap_timer</th>
<th>- [in]</th>
<th>MCPWM capture timer, allocated by mcpwm_new_capture_timer()</th>
</tr>
</thead>
<tbody>
<tr>
<td>返回</td>
<td>ESP_OK: Start MCPWM capture timer successfully</td>
<td></td>
<td>ESP_ERR_INVALID_ARG: Start MCPWM capture timer failed because of invalid argument</td>
</tr>
<tr>
<td></td>
<td>ESP_FAIL: Start MCPWM capture timer failed because of other error</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
esp_err_t mcpwm_capture_timer_stop (mcpwm_cap_timer_handle_t cap_timer)
```

Stop MCPWM capture timer.

<table>
<thead>
<tr>
<th>参数</th>
<th>cap_timer</th>
<th>- [in]</th>
<th>MCPWM capture timer, allocated by mcpwm_new_capture_timer()</th>
</tr>
</thead>
<tbody>
<tr>
<td>返回</td>
<td>ESP_OK: Stop MCPWM capture timer successfully</td>
<td></td>
<td>ESP_ERR_INVALID_ARG: Stop MCPWM capture timer failed because of invalid argument</td>
</tr>
<tr>
<td></td>
<td>ESP_FAIL: Stop MCPWM capture timer failed because of other error</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
esp_err_t mcpwm_capture_timer_get_resolution (mcpwm_cap_timer_handle_t cap_timer, uint32_t *out_resolution)
```

Get MCPWM capture timer resolution, in Hz.

<table>
<thead>
<tr>
<th>参数</th>
<th>cap_timer</th>
<th>- [in]</th>
<th>MCPWM capture timer, allocated by mcpwm_new_capture_timer()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>out_resolution</td>
<td>- [out]</td>
<td>Returned capture timer resolution, in Hz</td>
</tr>
<tr>
<td>返回</td>
<td>ESP_OK: Get capture timer resolution successfully</td>
<td></td>
<td>ESP_ERR_INVALID_ARG: Get capture timer resolution failed because of invalid argument</td>
</tr>
<tr>
<td></td>
<td>ESP_FAIL: Get capture timer resolution failed because of other error</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
esp_err_t mcpwm_capture_timer_set_phase_on_sync (mcpwm_cap_timer_handle_t cap_timer, const mcpwm_capture_timer_sync_phase_config_t *config)
```

Set sync phase for MCPWM capture timer.

<table>
<thead>
<tr>
<th>参数</th>
<th>cap_timer</th>
<th>- [in]</th>
<th>MCPWM capture timer, allocated by mcpwm_new_capture_timer()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>config</td>
<td>- [in]</td>
<td>MCPWM capture timer sync phase configuration</td>
</tr>
</tbody>
</table>
Chapter 2. API

ESP_OK: Set sync phase for MCPWM capture timer successfully
ESP_ERR_INVALID_ARG: Set sync phase for MCPWM capture timer failed because of invalid argument
ESP_FAIL: Set sync phase for MCPWM capture timer failed because of other error

```
esp_err_t mcpwm_new_capture_channel (mcpwm_cap_timer_handle_t cap_timer, const mcpwm_capture_channel_config_t *config, mcpwm_cap_channel_handle_t *ret_cap_channel)
```

Create MCPWM capture channel.

- `cap_timer` [in] MCPWM capture timer, allocated by mcpwm_new_capture_timer(), will be connected to the new capture channel
- `config` [in] MCPWM capture channel configuration
- `ret_cap_channel` [out] Returned MCPWM capture channel

ESP_OK: Create MCPWM capture channel successfully
ESP_ERR_INVALID_ARG: Create MCPWM capture channel failed because of invalid argument
ESP_ERR_NO_MEM: Create MCPWM capture channel failed because of other error
ESP_ERR_NOT_FOUND: Create MCPWM capture channel failed because can’t find free resource
ESP_FAIL: Create MCPWM capture channel failed because of other error

```
esp_err_t mcpwm_del_capture_channel (mcpwm_cap_channel_handle_t cap_channel)
```

Delete MCPWM capture channel.

- `cap_channel` [in] MCPWM capture channel handle, allocated by mcpwm_new_capture_channel()

ESP_OK: Delete MCPWM capture channel successfully
ESP_ERR_INVALID_ARG: Delete MCPWM capture channel failed because of invalid argument
ESP_FAIL: Delete MCPWM capture channel failed because of other error

```
esp_err_t mcpwm_capture_channel_enable (mcpwm_cap_channel_handle_t cap_channel)
```

Enable MCPWM capture channel.

- `cap_channel` [in] MCPWM capture channel handle, allocated by mcpwm_new_capture_channel()

ESP_OK: Enable MCPWM capture channel successfully
ESP_ERR_INVALID_ARG: Enable MCPWM capture channel failed because of invalid argument

备注：This function will transit the channel state from init to enable.

备注：This function will enable the interrupt service, if it’s lazy installed in mcpwm_capture_channel_register_event_callbacks().
• ESP_ERR_INVALID_STATE: Enable MCPWM capture channel failed because the channel is already enabled
• ESP_FAIL: Enable MCPWM capture channel failed because of other error

\texttt{esp_err_t mcpwm_capture_channel_disable(mcpwm_cap_channel_handle_t cap\_channel)}

Disable MCPWM capture channel.

参数 \texttt{cap\_channel} — [in] MCPWM capture channel handle, allocated by \texttt{mcpwm_new_capture_channel()}

返回
• ESP_OK: Disable MCPWM capture channel successfully
• ESP_ERR_INVALID_ARG: Disable MCPWM capture channel failed because of invalid argument
• ESP_ERR_INVALID_STATE: Disable MCPWM capture channel failed because the channel is not enabled yet
• ESP_FAIL: Disable MCPWM capture channel failed because of other error

\texttt{esp_err_t mcpwm_capture_channel_register_event_callbacks(mcpwm_cap_channel_handle_t cap\_channel, const mcpwm_capture_event_callbacks_t\* cbs, void* user\_data)}

Set event callbacks for MCPWM capture channel.

备注：The first call to this function needs to be before the call to \texttt{mcpwm_capture_channel_enable}

备注：User can deregister a previously registered callback by calling this function and setting the callback member in the \texttt{cbs} structure to NULL.

参数
• \texttt{cap\_channel} — [in] MCPWM capture channel handle, allocated by \texttt{mcpwm_new_capture_channel()}
• \texttt{cbs} — [in] Group of callback functions
• \texttt{user\_data} — [in] User data, which will be passed to callback functions directly

返回
• ESP_OK: Set event callbacks successfully
• ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
• ESP_ERR_INVALID_STATE: Set event callbacks failed because the channel is not in init state
• ESP_FAIL: Set event callbacks failed because of other error

\texttt{esp_err_t mcpwm_capture_channel_trigger_soft_catch(mcpwm_cap_channel_handle_t cap\_channel)}

Trigger a catch by software.

参数 \texttt{cap\_channel} — [in] MCPWM capture channel handle, allocated by \texttt{mcpwm_new_capture_channel()}

返回
• ESP_OK: Trigger software catch successfully
• ESP_ERR_INVALID_ARG: Trigger software catch failed because of invalid argument
• ESP_ERR_INVALID_STATE: Trigger software catch failed because the channel is not enabled yet
• ESP_FAIL: Trigger software catch failed because of other error

\textbf{Structures}
struct mcpwm_capture_timer_config_t
MCPWM capture timer configuration structure.

Public Members

int group_id
Specify from which group to allocate the capture timer

mcpwm_capture_clock_source_t clk_src
MCPWM capture timer clock source

struct mcpwm_capture_timer_sync_phase_config_t
MCPWM Capture timer sync phase configuration.

Public Members

mcpwm_sync_handle_t sync_src
The sync event source

uint32_t count_value
The count value that should lock to upon sync event

mcpwm_timer_direction_t direction
The count direction that should lock to upon sync event

struct mcpwm_capture_channel_config_t
MCPWM capture channel configuration structure.

Public Members

int gpio_num
GPIO used capturing input signal

uint32_t prescale
Prescale of input signal, effective frequency = cap_input_clk/prescale

uint32_t pos_edge
Whether to capture on positive edge

uint32_t neg_edge
Whether to capture on negative edge

uint32_t pull_up
Whether to pull up internally

uint32_t pull_down
Whether to pull down internally
uint32_t invert_cap_signal
  Invert the input capture signal

uint32_t io_loop_back
  For debug/test, the signal output from the GPIO will be fed to the input path as well

struct mcpwm_capture_channel_config_t::[anonymous] flags
  Extra configuration flags for capture channel

struct mcpwm_capture_event_callbacks_t
  Group of supported MCPWM capture event callbacks.

备注：The callbacks are all running under ISR environment

公共成员

mcpwm_capture_event_cb_t on_cap
  Callback function that would be invoked when capture event occurred

header file

• components/driver/include/driver/mcpwm_types.h

结构体

struct mcpwm_timer_event_data_t
  MCPWM timer event data.

公共成员

uint32_t count_value
  MCPWM timer count value

mcpwm_timer_direction_t direction
  MCPWM timer count direction

struct mcpwm_brake_event_data_t
  MCPWM brake event data.

struct mcpwm_fault_event_data_t
  MCPWM fault event data.

struct mcpwm_compare_event_data_t
  MCPWM compare event data.
Public Members

uint32_t compare_ticks
    Compare value

mcpwm_timer_direction_t direction
    Count direction

struct mcpwm_capture_event_data_t
    MCPWM capture event data.

Public Members

uint32_t cap_value
    Captured value

mcpwm_capture_edge_t cap_edge
    Capture edge

Type Definitions

typedef struct mcpwm_timer_t *mcpwm_timer_handle_t
    Type of MCPWM timer handle.

typedef struct mcpwm_oper_t *mcpwm_oper_handle_t
    Type of MCPWM operator handle.

typedef struct mcpwm_cmpr_t *mcpwm_cmpr_handle_t
    Type of MCPWM comparator handle.

typedef struct mcpwm_gen_t *mcpwm_gen_handle_t
    Type of MCPWM generator handle.

typedef struct mcpwm_fault_t *mcpwm_fault_handle_t
    Type of MCPWM fault handle.

typedef struct mcpwm_sync_t *mcpwm_sync_handle_t
    Type of MCPWM sync handle.

typedef struct mcpwm_cap_timer_t *mcpwm_cap_timer_handle_t
    Type of MCPWM capture timer handle.

typedef struct mcpwm_cap_channel_t *mcpwm_cap_channel_handle_t
    Type of MCPWM capture channel handle.

typedef bool (*mcpwm_timer_event_cb_t)(mcpwm_timer_handle_t timer, const mcpwm_timer_event_data_t *edata, void *user_ctx)
    MCPWM timer event callback function.

    Param timer [in] MCPWM timer handle
Chapter 2. API

**Param edata** [in] MCPWM timer event data, fed by driver
**Param user_ctx** [in] User data, set in `mcpwm_timer_register_eventcallbacks()`
**Return** Whether a high priority task has been waken up by this function

typedef bool (`mcpwm_brake_event_cb_t`)(`mcpwm_oper_handle_t` oper, const `mcpwm_brake_event_data_t` *edata, void *user_ctx)

MCPWM operator brake event callback function.

**Param oper** [in] MCPWM operator handle
**Param edata** [in] MCPWM brake event data, fed by driver
**Param user_ctx** [in] User data, set in `mcpwm_operator_register_eventcallbacks()`
**Return** Whether a high priority task has been waken up by this function

typedef bool (`mcpwm_fault_event_cb_t`)(`mcpwm_fault_handle_t` fault, const `mcpwm_fault_event_data_t` *edata, void *user_ctx)

MCPWM fault event callback function.

**Param fault** MCPWM fault handle
**Param edata** MCPWM fault event data, fed by driver
**Param user_ctx** User data, set in `mcpwm_fault_register_eventcallbacks()`
**Return** Whether a task switch is needed after the callback returns

typedef bool (`mcpwm_compare_event_cb_t`)(`mcpwm_cmpr_handle_t` comparator, const `mcpwm_compare_event_data_t` *edata, void *user_ctx)

MCPWM comparator event callback function.

**Param comparator** MCPWM comparator handle
**Param edata** MCPWM comparator event data, fed by driver
**Param user_ctx** User data, set in `mcpwm_comparator_register_eventcallbacks()`
**Return** Whether a high priority task has been waken up by this function

typedef bool (`mcpwm_capture_event_cb_t`)(`mcpwm_cap_channel_handle_t` cap_channel, const `mcpwm_capture_event_data_t` *edata, void *user_ctx)

MCPWM capture event callback function.

**Param cap_channel** MCPWM capture channel handle
**Param edata** MCPWM capture event data, fed by driver
**Param user_ctx** User data, set in `mcpwm_capture_channel_register_eventcallbacks()`
**Return** Whether a high priority task has been waken up by this function

**Header File**

- components/hal/include/hal/mcpwm_types.h

**Type Definitions**

typedef `soc_periph_mcpwm_timer_clk_src_t` `mcpwm_timer_clock_source_t`

MCPWM timer clock source.

typedef `soc_periph_mcpwm_capture_clk_src_t` `mcpwm_capture_clock_source_t`

MCPWM capture clock source.

**Enumerations**

enum `mcpwm_timer_direction_t`

MCPWM timer count direction.

**Values:**
enumerator MCPWM_TIMER_DIRECTION_UP
    Counting direction: Increase

enumerator MCPWM_TIMER_DIRECTION_DOWN
    Counting direction: Decrease

enum mcpwm_timer_event_t
    MCPWM timer events.
    Values:
    enumerator MCPWM_TIMER_EVENT_EMPTY
        MCPWM timer counts to zero (i.e. counter is empty)
    enumerator MCPWM_TIMER_EVENT_FULL
        MCPWM timer counts to peak (i.e. counter is full)
    enumerator MCPWM_TIMER_EVENT_INVALID
        MCPWM timer invalid event

enum mcpwm_timer_count_mode_t
    MCPWM timer count modes.
    Values:
    enumerator MCPWM_TIMER_COUNT_MODE_PAUSE
        MCPWM timer paused
    enumerator MCPWM_TIMER_COUNT_MODE_UP
        MCPWM timer counting up
    enumerator MCPWM_TIMER_COUNT_MODE_DOWN
        MCPWM timer counting down
    enumerator MCPWM_TIMER_COUNT_MODE_UP_DOWN
        MCPWM timer counting up and down

enum mcpwm_timer_start_stop_cmd_t
    MCPWM timer commands, specify the way to start or stop the timer.
    Values:
    enumerator MCPWM_TIMER_STOP_EMPTY
        MCPWM timer stops when next count reaches zero
    enumerator MCPWM_TIMER_STOP_FULL
        MCPWM timer stops when next count reaches peak
    enumerator MCPWM_TIMER_START_NO_STOP
        MCPWM timer starts couting, and don’t stop until received stop command
enumerator **MCPWM_TIMER_START_STOP_EMPTY**
MCPWM timer starts counting and stops when next count reaches zero

enumerator **MCPWM_TIMER_START_STOP_FULL**
MCPWM timer starts counting and stops when next count reaches peak

**enum mcpwm_generator_action_t**
MCPWM generator actions.
Values:

enumerator **MCPWM_GEN_ACTION_KEEP**
Generator action: Keep the same level

enumerator **MCPWM_GEN_ACTION_LOW**
Generator action: Force to low level

enumerator **MCPWM_GEN_ACTION_HIGH**
Generator action: Force to high level

enumerator **MCPWM_GEN_ACTION_TOGGLE**
Generator action: Toggle level

**enum mcpwm_operator_brake_mode_t**
MCPWM operator brake mode.
Values:

enumerator **MCPWM_OPER_BRAKE_MODE_CBC**
Brake mode: CBC (cycle by cycle)

enumerator **MCPWM_OPER_BRAKE_MODE_OST**
Brake mode, OST (one shot)

enumerator **MCPWM_OPER_BRAKE_MODE_INVALID**
MCPWM operator invalid brake mode

**enum mcpwm_capture_edge_t**
MCPWM capture edge.
Values:

enumerator **MCPWM_CAP_EDGE_POS**
Capture on the positive edge

enumerator **MCPWM_CAP_EDGE_NEG**
Capture on the negative edge
2.6.13 脉冲计数器 (PCNT)

概述
PCNT 用于统计输入信号的上升沿和/或下降沿的数量。ESP32 集成了多个脉冲计数单元，每个单元都是包含多个通道的独立计数器。通道可独立配置为统计上升沿或下降沿数量的递增计数器或递减计数器。
PCNT 通道可检测 边沿 信号及 电平 信号。对于比较简单的应用，检测边沿信号就足够了。PCNT 通道可检测上升沿信号、下降沿信号、同时也能设置为递增计数、递减计数，或停止计数。电平信号就是所谓的 控制信号，可用来控制边沿信号的计数模式。通过设置电平信号与边沿信号的检测模式，PCNT 单元可用作 正交解码器。
每个 PCNT 单元还包含一个滤波器，用于滤除线路毛刺。
PCNT 模块通常用于：
- 对一段时间内的脉冲计数，进而计算得到周期信号的频率；
- 对正交信号进行解码，进而获得速度和方向信号。

功能描述
PCNT 的功能从以下几个方面进行说明：
- 分配资源 - 说明如何通过配置分配 PCNT 单元和通道，以及在相应操作完成之后，如何回收单元和通道。
- 设置通道操作 - 说明如何设置通道针对不同信号沿和电平进行操作。
- PCNT 观察点 - 说明如何配置观察点，即当计数器达到某个数值时，命令 PCNT 单元触发某个事件。
- 注册事件回调函数 - 说明如何将您的代码挂载到观察点事件的回调函数上。
- 设置毛刺滤波器 - 说明如何使能毛刺滤波器并设置其时序参数。
- 使能和禁用单元 - 说明如何使能和关闭 PCNT 单元。
- 控制单元 IO 操作 - 说明 PCNT 单元的 IO 控制功能，例如使能毛刺滤波器，开启和停止 PCNT 单元，获取和清除计数。
- 电源管理 - 说明那些功能会阻止芯片进入低功耗模式。
- 支持 IRAM 安全中断 - 说明在缓存禁止的情况下，如何执行 PCNT 中断和 IO 控制功能。
- 支持线程安全 - 列出线程安全的 API。
- 支持的 Kconfig 选项 - 列出了支持的 Kconfig 选项，这些选项可实现不同的驱动效果。

分配资源 PCNT 单元和通道分别用 pcnt_unit_handle_t 与 pcnt_channel_handle_t 表示。所有的可用单元和通道都由驱动在资源池中进行维护，无需了解底层实例 ID。

安装 PCNT 单元 安装 PCNT 单元时，需要先完成配置 pcnt_unit_config_t:
- pcnt_unit_config_t::low_limit 与 pcnt_unit_config_t::high_limit 用于指定内部计数器的最小值和最大值。当计数器超过任一限值时，计数器将归零。
- pcnt_unit_config_t::accum_count 用于设置是否需要在硬件计数值溢出的时候进行累加保存。这有助于“拓宽”计数器的实际位宽。默认情况下，计数器的位宽最高只有 16 比特。请参考计数溢出补偿 了解如何利用此功能来补偿硬件计数器的溢出损失。
调用函数 pcnt_new_unit() 并将 pcnt_unit_config_t 作为其输入值，可对 PCNT 单元进行分配和初始化。该函数正常运行时，会返回一个 PCNT 单元句柄。没有可用的 PCNT 单元时，即 PCNT 单元全部被占用，该函数会返回错误 ESP_ERR_NOT_FOUND。可用的 PCNT 单元总数记录在 SOC_PCNT_UNITS_PER_GROUP 中，以供参考。

如果不再需要之前创建的某个 PCNT 单元，建议通过调用 pcnt_del_unit() 来回收该单元，从而该单元可用于其他用途。删除某个 PCNT 单元之前，需要满足以下条件：
- 该单元处于初始状态，即该单元要么已经被 pcnt_unit_disable() 禁用，要么尚未使能。

1 在不同的 ESP 芯片系列中，PCNT 单元和通道的数量可能会有差异。具体信息请参考 [TRM]，驱动不会禁止用户申请更多的 PCNT 单元和通道，但是当单元和通道资源全部被占用时，再调用单元和通道会返回错误。因此分配资源时，应注意检查返回值。如 pcnt_new_unit()。
安装 PCNT 通道 安装 PCNT 通道时，需要先初始化 pcnt_chan_config_t，然后调用 pcnt_new_channel()。对 pcnt_chan_config_t 配置如下所示:

- pcnt_chan_config_t::edge_gpio_num 与 pcnt_chan_config_t::level_gpio_num 用于指定边沿信号的 GPIO 编号。请注意，这两个参数未被使用时，可以设置为 -1，即成为虚拟 IO。对于一些简单的脉冲计数应用，电平信号或边沿信号是固定的（即不会发生改变），可将其设置为虚拟 IO，然后该信号会被连接到一个固定的高/低电平电平，这样就可以在通道分配时回收一个 GPIO。节省一个 GPIO 管脚资源。
- pcnt_chan_config_t::virt_edge_io_level 与 pcnt_chan_config_t::virt_level_io_level 用于指定边沿信号和电平信号的虚拟 IO 电平，以保证这些控制信号处于初始化状态。请注意，只有在 pcnt_chan_config_t::edge_gpio_num 或 pcnt_chan_config_t::level_gpio_num 设置为 -1 时，这两个参数才有效。
- pcnt_chan_config_t::invert_edge_input 与 pcnt_chan_config_t::invert_level_input 用于确定信号在输入 PCNT 之前是否需要被翻转，信号翻转由 GPIO 矩阵（不是 PCNT 单元）执行。
- pcnt_chan_config_t::io_loop back 仅用于调试，它可以达成 GPIO 的输入和输出路径。这样，就可以通过调用用于同一 GPIO 上的函数 gpio_set_level() 来模拟脉冲信号。

调用函数 pcnt_new_channel()，将 pcnt_chan_config_t 作为输入值并调用 pcnt_new_unit() 返回的 PCNT 单元句柄，可对 PCNT 通道进行分配和初始化。如果该函数正常运行，会返回一个 PCNT 通道句柄。如果没有可用的 PCNT 通道（PCNT 通道资源全部被占用），该函数会返回错误 ESP_ERR_NOT_FOUND。可用的 PCNT 通道总数记录在 SOC_PCNT_CHANNELS_PER_UNIT，供参考。注意，为某个单元安装 PCNT 通道时，应确保该单元处于初始化状态，否则函数 pcnt_new_channel() 会返回错误 ESP_ERR_INVALID_STATE。

如果不再需要之前创建的某个 PCNT 通道，建议通过调用 pcnt_del_channel() 回收该通道，从而该通道可用于其他用途。

设置通道操作 当输入脉冲信号切换时，PCNT 通道会增加，减少或停止计数。边沿信号及电平信号可设置为不同的计数器操作。

- pcnt_channel_set_edge_action() 为输入到 pcnt_chan_config_t::edge_gpio_num 的信号上升沿和下降沿设置操作，pcnt_channel_edge_action_t 中列出了支持的操作。
- pcnt_channel_set_level_action() 为输入到 pcnt_chan_config_t::level_gpio_num 的信号高电平和低电平设置操作，pcnt_channel_level_action_t 中列出了支持的操作。使用 pcnt_new_channel() 分配 PCNT 通道时，如果 pcnt_chan_config_t::level_gpio_num 被设置为 -1，就无需对该函数进行设置了。
// decrease the counter on rising edge, increase the counter on falling edge
ESP_ERROR_CHECK(pcnt_channel_set_edge_action(pcnt_chan, PCNT_CHANNEL_EDGE_ACTION_¬DECREASE, PCNT_CHANNEL_EDGE_ACTION_ACTION_INCREASE));
// keep the counting mode when the control signal is high level, and reverse the_¬counting mode when the control signal is low level
ESP_ERROR_CHECK(pcnt_channel_set_level_action(pcnt_chan, PCNT_CHANNEL_LEVEL_ACTION_¬KEEP, PCNT_CHANNEL_LEVEL_ACTION_ACTION_INVERSE));

PCNT 观察点 PCNT 单元可被设置为观察几个特定的数值，这些被观察的数值被称为观察点。观察点不能超过 pcnt_unit_config_t 设置的范围，最小值和最大值分别为 pcnt_unit_config_t::low_limit 和 pcnt_unit_config_t::high_limit。当计数器到达任一观察点时，会触发一个观察事件，如果在 pcnt_unit_register_event_callbacks() 注册过事件回调函数，该事件就会通过中断通知您。关于如何注册事件回调函数，请参考注册事件回调函数。

观察点分别可以通过 pcnt_unit_add_watch_point() 和 pcnt_unit_remove_watch_point() 进行添加和删除。常用的观察点包括过零、最大/最小计数以及其他阈值。可用的观察点是有限的，如果 pcnt_unit_add_watch_point() 无法获得空闲硬件资源来存储观察点，会返回错误 ESP_ERR_NOT_FOUND。不能多次添加同一个观察点，否则将返回错误 ESP_ERR_INVALID_STATE。

建议通过 pcnt_unit_remove_watch_point() 删除未使用的观察点来回收资源。

备注：由于硬件上的限制，在添加一个新的观察点后，你需要调用 pcnt_unit_clear_count() 函数来使之生效。

注册事件回调函数 当 PCNT 单元的数值达到任一使用的观察点的数值时，会触发相应的事件并通知 CPU。如果您想在事件触发时执行相关函数，可通过调用 pcnt_unit_register_event_callbacks() 将函数挂载到中断服务程序 (ISR) 上。

pcnt_event_callbacks_t 列出了所有支持的事件回调函数：
• pcnt_event_callbacks_t::on_reach 用于为观察点事件设置回调函数。由于该回调函数是在 ISR 的上下文中被调用的，必须确保该函数不会阻塞调用的任务。(例如，可确保只有以 ISR 为后缀的 FreeRTOS API 才能在函数中调用。) pcnt_watch_cb_t 中声明了该回调函数的原型。

可使用 user_ctx 将函数上下文保存到 pcnt_unit_register_event_callbacks() 中，这些数据会直接传递给回调函数。

驱动程序会将特定事件的数据写入回调函数中，例如，观察点事件数据被声明为 pcnt_watch_event_data_t：
• pcnt_watch_event_data_t::watch_point_value 用于保存触发该事件的观察点数值。
• pcnt_watch_event_data_t::zero_cross_mode 用于保存上一次 PCNT 单元的过零模式，pcnt_unit_zero_cross_mode_t 中列出了所有可能的过零模式，通常，不同的过零模式意味着不同的计数方向和计数步长。

注册回调函数会导致中断服务延迟安装，因此回调函数只能在 PCNT 单元被 pcnt_unit_enable() 使能之前调用，否则，回调函数会返回错误 ESP_ERR_INVALID_STATE。

static bool example_pcnt_on_reach(pcnt_unit_handle_t unit, const pcnt_watch_event_¬data_t *data, void *user_ctx)
{
    BaseType_t high_task_wakeup;
    QueueHandle_t queue = (QueueHandle_t)user_ctx;
    // send watch point to queue, from this interrupt callback
}
xQueueSendFromISR(queue, &edata->watch_point_value), &high_task_wakeup);
// return whether a high priority task has been waken up by this function
return (high_task_wakeup == pdTRUE);
}

pcnt_event_callbacks_t cbs = {
    .on_reach = example_pcnt_on_reach,
};
QueueHandle_t queue = xQueueCreate(10, sizeof(int));
ESP_ERROR_CHECK(pcnt_unit_register_event_callbacks(pcnt_unit, &cbs, queue));

### 设置毛刺滤波器
PCNT单元的滤波器可滤除信号中的短时毛刺。`pcnt_glitch_filter_config_t`中列出了毛刺滤波器的配置参数：

- `pcnt_glitch_filter_config_t::max_glitch_ns`设置最大的毛刺宽度。单位为微秒。如果一个信号脉冲的宽度小于该数值，则该信号会被认定为噪声而不会触发计数器操作。

可通过调用`pcnt_unit_set_glitch_filter()`来使能毛刺滤波器，并对上述参数进行配置。之后，还可通过调用`pcnt_unit_set_glitch_filter()`来关闭毛刺滤波器，并将上述参数设置为`NULL`。

调用该函数时，PCNT单元应处于初始状态。否则，函数将返回错误`ESP_ERR_INVALID_STATE`。

```c
pcnt_glitch_filter_config_t filter_config = {
    .max_glitch_ns = 1000,
};
ESP_ERROR_CHECK(pcnt_unit_set_glitch_filter(pcnt_unit, &filter_config));
```

### 使能和禁用单元
在对PCNT单元进行IO控制之前，需要通过调用函数`pcnt_unit_enable()`来使能该PCNT单元。该函数将完成以下操作：

- 将PCNT单元的驱动状态从初始切换为使能。
- 如果中断服务已经在`pcnt_unit_register_event_callbacks()`延迟安装，使能中断服务。
- 如果电源管理锁已经在`pcnt_unit_set_glitch_filter()`延迟安装，获取该电源管理锁。请参考电源管理获取更多信息。

调用函数`pcnt_unit_disable()`会进行相反的操作，即PCNT单元的驱动状态切换回初始状态，禁用中断服务并释放电源管理锁。

### 控制单元IO操作

#### 启用/停用及清零
通过调用`pcnt_unit_start()`可启用PCNT单元，根据不同脉冲信号进行递增或递减计数；通过调用`pcnt_unit_stop()`可停用PCNT单元，当前的计数值会保留；通过调用`pcnt_unit_clear_count()`可将计数器清零。

注意`pcnt_unit_start()`和`pcnt_unit_stop()`应该在PCNT单元被`pcnt_unit_enable()`使能后调用，否则将返回错误`ESP_ERR_INVALID_STATE`。
获取计数器数值  
调用 `pcnt_unit_get_count()` 可随时获取当前计数器的数值。返回的计数值是一个带符号的整型数，其符号反映了计数的方向。

```c
int pulse_count = 0;
ESP_ERROR_CHECK(pcnt_unit_get_count(pcnt_unit, &pulse_count));
```

计数溢出补偿  
PCNT 内部的硬件计数器会在计数达到高/低门限时自动清零。如果你想补偿该计数值的溢出损失，以期进一步拓宽计数器的实际位宽，你可以：

1. 在安装 PCNT 计数单元的时候使能 `pcnt_unit_config_t::accum_count` 选项。
2. 将高/低计数门限设置为 `PCNT` 观察点。
3. 现在，`pcnt_unit_get_count()` 函数返回的计数值就会包含硬件计数器当前的计数值，累加上计数器溢出造成的损失。

备注：`pcnt_unit_clear_count()` 会复位该软件累加器。

电源管理  
使能电源管理（即 `CONFIG_PM_ENABLE` 开启）后，在进入 Light-sleep 模式之前，系统会调整 APB 的频率。这会改变 PCNT 毛刺滤波器的参数，从而可能导致有效信号被滤除。

驱动通过获取 `ESP_PM_APB_FREQ_MAX` 类型的电源管理锁来防止系统修改 APB 频率。每当通过 `pcnt_unit_set_glitch_filter()` 使能毛刺滤波器时，驱动可以保证系统在 `pcnt_unit_enable()` 使能 PCNT 单元后获取电源管理锁。而系统调用 `pcnt_unit_disable()` 之后，驱动会释放电源管理锁。

支持 IRAM 安全中断  
当缓存由于写入/擦除 flash 等原因被禁用时，PCNT 中断会默认被延迟。这会导致报警中断无法及时执行，从而无法满足实时性应用的要求。

Konfig 选项 `CONFIG_PCNT_ISR_IRAM_SAFE` 可以实现以下功能：

1. 即使缓存被禁用也可以使能中断服务
2. 将 ISR 使用的所有函数都放入 IRAM 中
3. 将驱动对象放入 DRAM（防止驱动对象被意外映射到 PSRAM 中）

这样，在缓存被禁用时，中断也可运行，但是这也会增加 IRAM 的消耗。

另外一个 Konfig 选项 `CONFIG_PCNT_CTRL_FUNC_IN_IRAM` 也可以把常用的 IO 控制函数放在 IRAM 中。这样，当缓存禁用时，这些函数仍然可以执行。这些 IO 控制函数如下所示：

```
• pcnt_unit_start()
• pcnt_unit_stop()
• pcnt_unit_clear_count()
• pcnt_unit_get_count()
```

支持线程安全  
驱动保证工厂函数 `pcnt_new_unit()` 与 `pcnt_new_channel()` 是线程安全的，因此您可以从 RTOS 任务中调用这些函数而无需使用额外的电源管理锁。以下函数可以在 ISR 上下文中运行，驱动可以防止这些函数在任务和 ISR 中同时被调用。

```
• pcnt_unit_start()
• pcnt_unit_stop()
• pcnt_unit_clear_count()
• pcnt_unit_get_count()
```

其他以 `pcnt_unit_handle_t` 和 `pcnt_channel_handle_t` 作为第一个参数的函数被视为线程不安全函数。在多任务场景下应避免调用这些函数。

```
• pcnt_event_callbacks_t::on_reach
```

2 `pcnt_event_callbacks_t::on_reach` 回调函数和其调用的函数也应该放在 IRAM 中。
支持的 Kconfig 选项

- `CONFIG_PCNT_CTRL_FUNC_IN_IRAM` 用于确定 PCNT 控制函数的位置 (放在 IRAM 还是 flash 中)，请参考支持 IRAM 安全中断 获取更多信息。
- `CONFIG_PCNT_ISR_IRAM_SAFE` 用于控制当缓存禁用时，默认的 ISR 姿构是否可以工作，请参考支持 IRAM 安全中断 获取更多信息。
- `CONFIG_PCNT_ENABLE_DEBUG_LOG` 用于使能调试日志输出，而这一会增大固件二进制文件。

应用实例

- 对旋转编码器的正转信号进行解码的实例请参考：peripherals/pcnt/rotary_encoder。

API 参考

Header File

- components/driver/include/driver/pulse_cnt.h

Functions

```c
esp_err_t pcnt_new_unit (const pcnt_unit_config_t *config, pcnt_unit_handle_t *ret_unit)
```

Create a new PCNT unit, and return the handle.

备注: The newly created PCNT unit is put in the init state.

参数

- `config` [in] PCNT unit configuration
- `ret_unit` [out] Returned PCNT unit handle

返回

- ESP_OK: Create PCNT unit successfully
- ESP_ERR_INVALID_ARG: Create PCNT unit failed because of invalid argument (e.g. high/low limit value out of the range)
- ESP_ERR_NO_MEM: Create PCNT unit failed because out of memory
- ESP_ERR_NOT_FOUND: Create PCNT unit failed because all PCNT units are used up and no more free one
- ESP_FAIL: Create PCNT unit failed because of other error

```c
esp_err_t pcnt_del_unit (pcnt_unit_handle_t unit)
```

Delete the PCNT unit handle.

备注: A PCNT unit can’t be in the enable state when this function is invoked. See also `pcnt_unit_disable()` for how to disable a unit.

参数

- `unit` [in] PCNT unit handle created by `pcnt_new_unit()`

返回

- ESP_OK: Delete the PCNT unit successfully
- ESP_ERR_INVALID_ARG: Delete the PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete the PCNT unit failed because the unit is not in init state or some PCNT channel is still in working
- ESP_FAIL: Delete the PCNT unit failed because of other error

```c
esp_err_t pcnt_unit_set_glitch_filter (pcnt_unit_handle_t unit, const pcnt_glitch_filter_config_t *config)
```
Set glitch filter for PCNT unit.

备注: The glitch filter module is clocked from APB, and APB frequency can be changed during DFS, which in return make the filter out of action. So this function will lazy-install a PM lock internally when the power management is enabled. With this lock, the APB frequency won’t be changed. The PM lock can be uninstalled in `pcnt_del_unit()`.

备注: This function should be called when the PCNT unit is in the init state (i.e. before calling `pcnt_unit_enable()`)

参数
- `unit` [in] PCNT unit handle created by `pcnt_new_unit()`
- `config` [in] PCNT filter configuration, set config to NULL means disabling the filter function

返回
- ESP_OK: Set glitch filter successfully
- ESP_ERR_INVALID_ARG: Set glitch filter failed because of invalid argument (e.g. glitch width is too big)
- ESP_ERR_INVALID_STATE: Set glitch filter failed because the unit is not in the init state
- ESP_FAIL: Set glitch filter failed because of other error

`esp_err_t pcnt_unit_enable (pcnt_unit_handle_t unit)`
Enable the PCNT unit.

备注: This function will transit the unit state from init to enable.

备注: This function will enable the interrupt service, if it’s lazy installed in `pcnt_unit_register_event_callbacks()`.

备注: This function will acquire the PM lock if it’s lazy installed in `pcnt_unit_set_glitch_filter()`.

备注: Enable a PCNT unit doesn’t mean to start it. See also `pcnt_unit_start()` for how to start the PCNT counter.

参数 `unit` [in] PCNT unit handle created by `pcnt_new_unit()`

返回
- ESP_OK: Enable PCNT unit successfully
- ESP_ERR_INVALID_ARG: Enable PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable PCNT unit failed because the unit is already enabled
- ESP_FAIL: Enable PCNT unit failed because of other error

`esp_err_t pcnt_unit_disable (pcnt_unit_handle_t unit)`
Disable the PCNT unit.
This function will do the opposite work to the `pcnt_unit_enable()` function.

Disable a PCNT unit doesn’t mean to stop it. See also `pcnt_unit_stop()` for how to stop the PCNT counter.

**Parameters**
- `unit`: PCNT unit handle created by `pcnt_new_unit()`

**Return**
- `ESP_OK`: Disable PCNT unit successfully
- `ESP_ERR_INVALID_ARG`: Disable PCNT unit failed because of invalid argument
- `ESP_ERR_INVALID_STATE`: Disable PCNT unit failed because the unit is not enabled yet
- `ESP_FAIL`: Disable PCNT unit failed because of other error

**Declaration**
```c
esp_err_t pcnt_unit_start(pcnt_unit_handle_t unit)
```

Start the PCNT unit, the counter will start to count according to the edge and/or level input signals.

**Parameters**
- `unit`: PCNT unit handle created by `pcnt_new_unit()`

**Return**
- `ESP_OK`: Start PCNT unit successfully
- `ESP_ERR_INVALID_ARG`: Start PCNT unit failed because of invalid argument
- `ESP_ERR_INVALID_STATE`: Start PCNT unit failed because the unit is not enabled yet
- `ESP_FAIL`: Start PCNT unit failed because of other error

**Declaration**
```c
esp_err_t pcnt_unit_stop(pcnt_unit_handle_t unit)
```

Stop PCNT from counting.

**Parameters**
- `unit`: PCNT unit handle created by `pcnt_new_unit()`

**Return**
- `ESP_OK`: Stop PCNT unit successfully
- `ESP_ERR_INVALID_ARG`: Start PCNT unit failed because of invalid argument
- `ESP_ERR_INVALID_STATE`: Start PCNT unit failed because the unit is not enabled yet
- `ESP_FAIL`: Start PCNT unit failed because of other error
Chapter 2. API参考

参数 unit - [in] PCNT unit handle created by pcnt_new_unit()
返回
- ESP_OK: Stop PCNT unit successfully
- ESP_ERR_INVALID_ARG: Stop PCNT unit failed because of invalid argument
- ESP_ERR_INVALID_STATE: Stop PCNT unit failed because the unit is not enabled yet
- ESP_FAIL: Stop PCNT unit failed because of other error

**esp_err_t pcnt_unit_clear_count (pcnt_unit_handle_t unit)**
Clear PCNT pulse count value to zero.

注: It’s recommended to call this function after adding a watch point by pcnt_unit_add_watch_point(), so that the newly added watch point is effective immediately.

注: This function is allowed to run within ISR context

注: This function will be placed into IRAM if CONFIG_PCNT_CTRL_FUNC_IN_IRAM, so that it’s allowed to be executed when Cache is disabled

参数 unit - [in] PCNT unit handle created by pcnt_new_unit()
返回
- ESP_OK: Clear PCNT pulse count successfully
- ESP_ERR_INVALID_ARG: Clear PCNT pulse count failed because of invalid argument
- ESP_FAIL: Clear PCNT pulse count failed because of other error

**esp_err_t pcnt_unit_get_count (pcnt_unit_handle_t unit, int* value)**
Get PCNT count value.

注: This function is allowed to run within ISR context

注: This function will be placed into IRAM if CONFIG_PCNT_CTRL_FUNC_IN_IRAM, so that it’s allowed to be executed when Cache is disabled

参数
- unit - [in] PCNT unit handle created by pcnt_new_unit()
- value - [out] Returned count value
返回
- ESP_OK: Get PCNT pulse count successfully
- ESP_ERR_INVALID_ARG: Get PCNT pulse count failed because of invalid argument
- ESP_FAIL: Get PCNT pulse count failed because of other error

**esp_err_t pcnt_unit_register_event_callbacks (pcnt_unit_handle_t unit, const pcnt_event_callbacks_t *cbs, void *user_data)**
Set event callbacks for PCNT unit.

注: User registered callbacks are expected to be runnable within ISR context

注: The first call to this function needs to be before the call to pcnt_unit_enable
Chapter 2. API

[Chapter content]

**Note:** User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

**Function:** `pcnt_unit_add_watch_point`

Add a watch point for PCNT unit, PCNT will generate an event when the counter value reaches the watch point value.

**Parameters:**
- `unit` - [in] PCNT unit handle created by `pcnt_new_unit()`
- `watch_point` - [in] Value to be watched

**Return Values:**
- `ESP_OK`: Add watch point successfully
- `ESP_ERR_INVALID_ARG`: Add watch point failed because of invalid argument (e.g. the value to be watched is out of the limitation set in `pcnt_unit_config_t`)
- `ESP_ERR_INVALID_STATE`: Add watch point failed because the same watch point has already been added
- `ESP_ERR_NOT_FOUND`: Add watch point failed because no more hardware watch point can be configured
- `ESP_FAIL`: Add watch point failed because of other error

**Function:** `pcnt_unit_remove_watch_point`

Remove a watch point for PCNT unit.

**Parameters:**
- `unit` - [in] PCNT unit handle created by `pcnt_new_unit()`
- `watch_point` - [in] Watch point value

**Return Values:**
- `ESP_OK`: Remove watch point successfully
- `ESP_ERR_INVALID_ARG`: Remove watch point failed because of invalid argument
- `ESP_ERR_INVALID_STATE`: Remove watch point failed because the watch point was not added by `pcnt_unit_add_watch_point()` yet
- `ESP_FAIL`: Remove watch point failed because of other error

**Function:** `pcnt_new_channel`

Create PCNT channel for specific unit, each PCNT has several channels associated with it.

**Parameters:**
- `unit` - [in] PCNT unit handle created by `pcnt_new_unit()`
- `config` - [in] PCNT channel configuration
- `ret_chan` - [out] Returned channel handle

**Note:** This function should be called when the unit is in init state (i.e. before calling `pcnt_unit_enable()`)
• ESP_OK: Create PCNT channel successfully
• ESP_ERR_INVALID_ARG: Create PCNT channel failed because of invalid argument
• ESP_ERR_NO_MEM: Create PCNT channel failed because of insufficient memory
• ESP_ERR_NOT_FOUND: Create PCNT channel failed because all PCNT channels are used up and no more free one
• ESP_ERR_INVALID_STATE: Create PCNT channel failed because the unit is not in the init state
• ESP_FAIL: Create PCNT channel failed because of other error

Esp_err_t pcnt_del_channel (pcnt_channel_handle_t chan)
Delete the PCNT channel.

参数 chan - [in] PCNT channel handle created by pcnt_new_channel()

返回
• ESP_OK: Delete the PCNT channel successfully
• ESP_ERR_INVALID_ARG: Delete the PCNT channel failed because of invalid argument
• ESP_FAIL: Delete the PCNT channel failed because of other error

Esp_err_t pcnt_channel_set_edge_action (pcnt_channel_handle_t chan, pcnt_channel_edge_action_t pos_act, pcnt_channel_edge_action_t neg_act)
Set channel actions when edge signal changes (e.g. falling or rising edge occurred). The edge signal is input from the edge_gpio_num configured in pcnt_chan_config_t. We use these actions to control when and how to change the counter value.

参数
• chan - [in] PCNT channel handle created by pcnt_new_channel()
• pos_act - [in] Action on posedge signal
• neg_act - [in] Action on negedge signal

返回
• ESP_OK: Set edge action for PCNT channel successfully
• ESP_ERR_INVALID_ARG: Set edge action for PCNT channel failed because of invalid argument
• ESP_FAIL: Set edge action for PCNT channel failed because of other error

Esp_err_t pcnt_channel_set_level_action (pcnt_channel_handle_t chan, pcnt_channel_level_action_t high_act, pcnt_channel_level_action_t low_act)
Set channel actions when level signal changes (e.g. signal level goes from high to low). The level signal is input from the level_gpio_num configured in pcnt_chan_config_t. We use these actions to control when and how to change the counting mode.

参数
• chan - [in] PCNT channel handle created by pcnt_new_channel()
• high_act - [in] Action on high level signal
• low_act - [in] Action on low level signal

返回
• ESP_OK: Set level action for PCNT channel successfully
• ESP_ERR_INVALID_ARG: Set level action for PCNT channel failed because of invalid argument
• ESP_FAIL: Set level action for PCNT channel failed because of other error

Structures

struct pcnt_watch_event_data_t
PCNT watch event data.

Public Members
int watch_point_value
    Watch point value that triggered the event

pcnt_unit_zero_cross_mode_t zero_cross_mode
    Zero cross mode

struct pcnt_event_callbacks_t
    Group of supported PCNT callbacks.

备注: The callbacks are all running under ISR environment

备注: When CONFIG_PCNT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM.

Public Members

pcnt_watch_cb_t on_reach
    Called when PCNT unit counter reaches any watch point

struct pcnt_unit_config_t
    PCNT unit configuration.

Public Members

int low_limit
    Low limitation of the count unit, should be lower than 0

int high_limit
    High limitation of the count unit, should be higher than 0

uint32_t accum_count
    Whether to accumulate the count value when overflows at the high/low limit

struct pcnt_unit_config_t::[anonymous] flags
    Extra flags

struct pcnt_chan_config_t
    PCNT channel configuration.

Public Members

int edge_gpio_num
    GPIO number used by the edge signal, input mode with pull up enabled. Set to -1 if unused
int level_gpio_num
    GPIO number used by the level signal, input mode with pull up enabled. Set to -1 if unused

uint32_t invert_edge_input
    Invert the input edge signal

uint32_t invert_level_input
    Invert the input level signal

uint32_t virt_edge_io_level
    Virtual edge IO level, 0: low, 1: high. Only valid when edge_gpio_num is set to -1

uint32_t virt_level_io_level
    Virtual level IO level, 0: low, 1: high. Only valid when level_gpio_num is set to -1

uint32_t io_loop_back
    For debug/test, the signal output from the GPIO will be fed to the input path as well

struct pcnt_chan_config_t::[anonymous] flags
    Channel config flags

struct pcnt_glitch_filter_config_t
    PCNT glitch filter configuration.

Public Members

uint32_t max_glitch_ns
    Pulse width smaller than this threshold will be treated as glitch and ignored, in the unit of ns

Type Definitions

typedef struct pcnt_unit_t *pcnt_unit_handle_t
    Type of PCNT unit handle.

typedef struct pcnt_chan_t *pcnt_channel_handle_t
    Type of PCNT channel handle.

typedef bool (*)(pcnt_unit_handle_t unit, const pcnt_watch_event_data_t *edata, void *user_ctx) pcnt_watch_cb_t
    PCNT watch event callback prototype.

备注： The callback function is invoked from an ISR context, so it should meet the restrictions of not calling
any blocking APIs when implementing the callback. e.g. must use ISR version of FreeRTOS APIs.

Param unit [in] PCNT unit handle
Param edata [in] PCNT event data, fed by the driver
Param user_ctx [in] User data, passed from pcnt_unit_register_event_callbacks()
Return Whether a high priority task has been woken up by this function
Header File

- components/hal/include/hal/pcnt_types.h

Enumerations

```c
enum pcnt_channel_level_action_t
    PCNT channel action on control level.

Values:

enumerator PCNT_CHANNEL_LEVEL_ACTION_KEEP
    Keep current count mode

enumerator PCNT_CHANNEL_LEVEL_ACTION_INVERSE
    Invert current count mode (increase -> decrease, decrease -> increase)

enumerator PCNT_CHANNEL_LEVEL_ACTION_HOLD
    Hold current count value
```

```c
enum pcnt_channel_edge_action_t
    PCNT channel action on signal edge.

Values:

enumerator PCNT_CHANNEL_EDGE_ACTION_HOLD
    Hold current count value

enumerator PCNT_CHANNEL_EDGE_ACTION_INCREASE
    Increase count value

enumerator PCNT_CHANNEL_EDGE_ACTION_DECREASE
    Decrease count value
```

```c
enum pcnt_unit_zero_cross_mode_t
    PCNT unit zero cross mode.

Values:

enumerator PCNT_UNIT_ZERO_CROSS_POS_ZERO
    start from positive value, end to zero, i.e. +N->0

enumerator PCNT_UNIT_ZERO_CROSS_NEGZERO
    start from negative value, end to zero, i.e. -N->0

enumerator PCNT_UNIT_ZERO_CROSS_NEG_POS
    start from negative value, end to positive value, i.e. -N->+M

enumerator PCNT_UNIT_ZERO_CROSS_POS_NEG
    start from positive value, end to negative value, i.e. +N->-M
```
2.6.14 Remote Control Transceiver (RMT)

Introduction

The RMT (Remote Control Transceiver) peripheral was designed to act as an infrared transceiver. However, due to the flexibility of its data format, the functionality of RMT can be extended to a versatile and general purpose transceiver. From the perspective of network layering, the RMT hardware contains both physical and data link layer. The physical layer defines the communication media and bit signal representation. The data link layer defines the format of an RMT frame. The minimal data unit in the frame is called RMT symbol, which is represented by \texttt{rmt_symbol_word_t} in the driver.

ESP32 contains multiple channels in the RMT peripheral. Each channel can be configured as either transmitter or receiver, independently.

Typically, the RMT peripheral can be used in the following scenarios:

- Transmit or receive infrared signals, with any IR protocols, e.g. NEC
- General purpose sequence generator
- Transmit signals in a hardware controlled loop, with finite or infinite number of times
- Multi-channel simultaneous transmission
- Modulate the carrier to the output signal or demodulate the carrier from the input signal

Layout of RMT Symbols

The RMT hardware defines data in its own pattern—the RMT symbol. Each symbol consists of two pairs of two values. The first value in a pair describes the signal duration in RMT ticks and is 15 bits long. The second provides the signal level (high or low) and is contained in a single bit, as shown below:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Structure of RMT symbols (L - signal level)](image)

RMT Transmitter Overview

The data path and control path of an RMT TX channel is illustrated in the figure below:

The driver will encode user’s data into RMT data format, then the RMT transmitter can generate the waveforms according to the encoding artifacts. It is also possible to modulate a high frequency carrier signal before being routed to a GPIO pad.

RMT Receiver Overview

The data path and control path of an RMT RX channel is illustrated in the figure below:

The RMT receiver can sample incoming signals into RMT data format, and store the data in memory. It’s feasible to tell the receiver the basic characteristics of the incoming signal, so that the signal’s stop condition can be recognized, and signal glitches and noise can be filtered out. The RMT peripheral also supports demodulating the high frequency carrier from the base signal.

1 Different ESP chip series might have different number of RMT channels. Please refer to the [TRM] for details. The driver won’t forbid you from applying for more RMT channels, but it will return error when there’s no hardware resources available. Please always check the return value when doing Resource Allocation.
图 14: RMT Transmitter Overview

图 15: RMT Receiver Overview
Chapter 2. API

Functional Overview

Description of the RMT functionality is divided into the following sections:

- **Resource Allocation** - covers how to allocate RMT channels with properly set of configurations. It also covers how to recycle the resources when they finished working.
- **Carrier Modulation and Demodulation** - describes how to modulate carrier for TX channel and demodulate carrier for RX channel.
- **Register Event Callbacks** - covers how to hook user specific code to RMT channel specific events.
- **Enable and Disable channel** - shows how to enable and disable the RMT channel.
- **Initiate TX Transaction** - describes the steps to initiate a transaction for TX channel.
- **Initiate RX Transaction** - describes the steps to initiate a transaction for RX channel.
- **Multiple Channels Simultaneous Transmission** - describes how to collect multiple channels into a sync group and start transaction at the same time.
- **RMT Encoder** - focuses on how to write a customized encoder in a combination way, with the help of the primitive encoders provided by the driver.
- **Power Management** - describes how different source clock will affect power consumption.
- **IRAM Safe** - describes tips on how to make the RMT interrupt work better along with a disabled cache.
- **Thread Safety** - lists which APIs are guaranteed to be thread safe by the driver.
- **Kconfig Options** - lists the supported Kconfig options that can bring different effects to the driver.

**Resource Allocation**

Both RMT TX and RX channels are represented by `rmt_channel_handle_t` in the driver. The available channels are managed in a resource pool, which will hand out a free channel on request.

**Install RMT TX Channel**

To install an RMT TX channel, there’s a configuration structure that needs to be given in advance: `rmt_tx_channel_config_t`:

- `rmt_tx_channel_config_t::gpio_num` sets the GPIO number used by the transmitter.
- `rmt_tx_channel_config_t::clk_src` selects the source clock for the RMT channel. The available clocks are listed in `rmt_clock_source_t`. Note that, the selected clock will also be used by other channels, which means user should ensure this configuration is same when allocating other channels, regardless of TX or RX. For the effect on power consumption of different clock source, please refer to **Power Management** section.
- `rmt_tx_channel_config_t::resolution_hz` sets the resolution of the internal tick counter. The timing parameter of RMT signal is calculated based on this tick.
- `rmt_tx_channel_config_t::mem_block_symbols` sets the size of the dedicated memory block or DMA buffer that is used to store RMT encoding artifacts.
- `rmt_tx_channel_config_t::trans_queue_depth` sets the depth of internal transaction queue, the deeper the queue, the more transactions can be prepared in the backlog.
- `rmt_tx_channel_config_t::invert_out` is used to decide whether to invert the RMT signal before sending it to the GPIO pad.
- `rmt_tx_channel_config_t::with_dma` is used to indicate if the channel needs a DMA backend. A channel with DMA attached can offload the CPU by a lot. However, DMA backend is not available on all ESP chips, please refer to [TRM] before you enable this option. Or you might encounter `ESP_ERR_NOT_SUPPORTED` error.
- `rmt_tx_channel_config_t::io_loop_back` enables both the GPIO’ s input and output ability through the GPIO matrix peripheral. Meanwhile, if both TX and RX channels are bound to the same GPIO, then monitoring of the data transmission line can be realized.
- `rmt_tx_channel_config_t::io_od_mode` configures the GPIO as open-drain mode. It is useful for simulating bi-directional buses, such as 1-wire bus, combined with `rmt_tx_channel_config_t::io_loop_back`.

Once the `rmt_tx_channel_config_t` structure is populated with mandatory parameters, users can call `rmt_new_tx_channel()` to allocate and initialize a TX channel. This function will return an RMT channel handle if it runs correctly. Specifically, when there are no more free channels in the RMT resource pool, this function will return `ESP_ERR_NOT_FOUND` error. If some feature (e.g. DMA backend) is not supported by hardware, it will return `ESP_ERR_NOT_SUPPORTED` error.
Chapter 2. API

rmt_channel_handle_t tx_chan = NULL;
rmt_tx_channel_config_t tx_chan_config = {
    .clk_src = RMT_CLK_SRC_DEFAULT,  // select source clock
    .gpio_num = 0,                   // GPIO number
    .mem_block_symbols = 64,         // memory block size, 64 * 4 = 256Bytes
    .resolution_hz = 1 * 1000 * 1000, // 1MHz tick resolution, i.e. 1 tick = 1us
    .trans_queue_depth = 4,          // set the number of transactions that can... pend in the background
    .flags.invert_out = false,       // don't invert output signal
    .flags.with_dma = false,         // don't need DMA backend
};
ESP_ERROR_CHECK(rmt_new_tx_channel(&tx_chan_config, &tx_chan));

Install RMT RX Channel To install an RMT RX channel, there’s a configuration structure that needs to be given in advance: rmt_rx_channel_config_t:

- rmt_rx_channel_config_t::gpio_num sets the GPIO number used by the receiver.
- rmt_rx_channel_config_t::clk_src selects the source clock for the RMT channel. The available clocks are listed in rmt_clock_source_t. Note that, the selected clock will also be used by other channels, which means user should ensure this configuration is same when allocating other channels, regardless of TX or RX. For the effect on power consumption of different clock source, please refer to Power Management section.
- rmt_rx_channel_config_t::resolution_hz sets the resolution of the internal tick counter. The timing parameter of RMT signal is calculated based on this tick.
- rmt_rx_channel_config_t::mem_block_symbols sets the size of the dedicated memory block or DMA buffer that used to store RMT encoding artifacts.
- rmt_rx_channel_config_t::invert_in is used to decide whether to invert the input signals before they going into RMT receiver. The inversion is done by GPIO matrix instead of by the RMT peripheral.
- rmt_rx_channel_config_t::with_dma is used to indicate if the channel needs a DMA backend. A channel with DMA attached can offload the CPU by a lot. However, DMA backend is not available on all ESP chips, please refer to [TRM] before you enable this option. Or you might encounter ESP_ERR_NOT_SUPPORTED error.
- rmt_rx_channel_config_t::io_loop_back is for debugging purposes only. It enables both the GPIO’s input and output ability through the GPIO matrix peripheral. Meanwhile, if both TX and RX channels are bound to the same GPIO, then monitoring of the data transmission line can be realized.

Once the rmt_rx_channel_config_t structure is populated with mandatory parameters, users can call rmt_new_rx_channel() to allocate and initialize a RX channel. This function will return an RMT channel handle if it runs correctly. Specifically, when there are no more free channels in the RMT resource pool, this function will return ESP_ERR_NOT_FOUND error. If some feature (e.g. DMA backend) is not supported by hardware, it will return ESP_ERR_NOT_SUPPORTED error.

Uninstall RMT Channel If a previously installed RMT channel is no longer needed, it’s recommended to recycle the resources by calling rmt_del_channel(), which in return allows the underlying hardware to be usable for other purposes.
Carrying Modulation and Demodulation. The RMT transmitter can generate a carrier wave and modulate it onto the base signal. Compared to the base signal, the carrier frequency is usually high. In addition, the user can only set the frequency and duty cycle for the carrier. The RMT receiver can demodulate the carrier from the incoming signal. Note that, carrier modulation and demodulation is not supported on all ESP chips, please refer to [TRM] before configuring the carrier, or you might encounter a ESP_ERR_NOT_SUPPORTED error.

Carrier related configurations lie in rmt_carrier_config_t:

- rmt_carrier_config_t::frequency_hz sets the carrier frequency, in Hz.
- rmt_carrier_config_t::duty_cycle sets the carrier duty cycle.
- rmt_carrier_config_t::polarity_active_low sets the carrier polarity, i.e. on which level the carrier is applied.
- rmt_carrier_config_t::always_on sets whether to output the carrier even when the data transmission has finished. This configuration is only valid for TX channel.

For RX channel, we shouldn’t set the carrier frequency exactly to the theoretical value. It’s recommended to leave a tolerance for the carrier frequency. For example, in the snippet below, we set the frequency to 25KHz, instead of the 38KHz that configured on the TX side. The reason is that reflection and refraction will occur when a signal travels through the air, leading to a distortion on the receiver side.

```c
rmt_carrier_config_t tx_carrier_cfg = {
    .duty_cycle = 0.33, // duty cycle 33%
    .frequency_hz = 38000, // 38KHz
    .flags.polarity_active_low = false, // carrier should modulated to high level
};
// modulate carrier to TX channel
ESP_ERROR_CHECK(rmt_apply_carrier(tx_chan, &tx_carrier_cfg));

rmt_carrier_config_t rx_carrier_cfg = {
    .duty_cycle = 0.33, // duty cycle 33%
    .frequency_hz = 25000, // 25KHz carrier, should be smaller than...
    .transmitter's carrier frequency
    .flags.polarity_active_low = false, // the carrier is modulated to high level
};
// demodulate carrier from RX channel
ESP_ERROR_CHECK(rmt_apply_carrier(rx_chan, &rx_carrier_cfg));
```

Register Event Callbacks. When an RMT channel finishes transmitting or receiving, a specific event will be generated and notify the CPU by interrupt. If you have some function that needs to be called when those events occurred, you can hook your function to the ISR (Interrupt Service Routine) by calling rmt_tx_register_event_callbacks() and rmt_rx_register_event_callbacks() for TX and RX channel respectively. Since the registered callback functions are called in the interrupt context, user should ensure the callback function doesn’t attempt to block (e.g. by making sure that only FreeRTOS APIs with ISR suffix are called from within the function). The callback function has a boolean return value, to tell the caller whether a high priority task is woke up by it.

TX channel supported event callbacks are listed in the rmt_tx_event_callbacks_t:

- rmt_tx_event_callbacks_t::on_trans_done sets a callback function for trans done event. The function prototype is declared in rmt_tx_done_callback_t.

RX channel supported event callbacks are listed in the rmt_rx_event_callbacks_t:

- rmt_rx_event_callbacks_t::on_recv_done sets a callback function for receive complete event. The function prototype is declared in rmt_rx_done_callback_t.

User can save own context in rmt_tx_register_event_callbacks() and rmt_rx_register_event_callbacks() as well, via the parameter user_data. The user data will be directly passed to each callback function.
In the callback function, users can fetch the event specific data that is filled by the driver in the edata. Note that the edata pointer is only valid for the duration of the callback.

The TX done event data is defined in `rmt_tx_done_event_data_t`:

- `rmt_tx_done_event_data_t::num_symbols` tells the number of transmitted RMT symbols. This also reflects the size of encoding artifacts.

The RX complete event data is defined in `rmt_rx_done_event_data_t`:

- `rmt_rx_done_event_data_t::received_symbols` points to the received RMT symbols. These symbols are saved in the buffer parameter of `rmt_receive()` function. User shouldn’t free this receive buffer before the callback returns.
- `rmt_rx_done_event_data_t::num_symbols` tells the number of received RMT symbols. This value won’t be bigger than buffer_size parameter of `rmt_receive()` function. If the buffer_size is not sufficient to accommodate all the received RMT symbols, the driver will truncate it.

Enable and Disable channel  
`rmt_enable()` must be called in advanced before transmitting or receiving RMT symbols. For transmitters, enabling a channel will enable a specific interrupt and prepare the hardware to dispatch transactions. For RX channels, enabling a channel will enable an interrupt, but the receiver is not started during this time, as it has no idea about the characteristics of the incoming signals. The receiver will be started in `rmt_receive()`.

`rmt_disable()` does the opposite work by disabling the interrupt and clearing pending status. The transmitter and receiver will be disabled as well.

```c
ESP_ERROR_CHECK(rmt_enable(tx_chan));
ESP_ERROR_CHECK(rmt_enable(rx_chan));
```

Initiate TX Transaction  
RMT is a special communication peripheral as it’s unable to transmit raw byte streams like SPI and I2C. RMT can only send data in its own format `rmt_symbol_word_t`. However, the hardware doesn’t help to convert the user data into RMT symbols, this can only be done in software — by the so-called **RMT Encoder**. The encoder is responsible for encoding user data into RMT symbols and then write to RMT memory block or DMA buffer. For how to create an RMT encoder, please refer to [RMT Encoder].

Once we got an encoder, we can initiate a TX transaction by calling `rmt_transmit()`. This function takes several positional parameters like channel handle, encoder handle, payload buffer. Besides that, we also need to provide a transmission specific configuration in `rmt_transmit_config_t`:

- `rmt_transmit_config_t::loop_count` sets the number of transmission loop. After the transmitter finished one round of transmission, it can restart the same transmission again if this value is not set to zero. As the loop is controlled by hardware, the RMT channel can be used to generate many periodic sequences at the cost of a very little CPU intervention. Specially, setting `rmt_transmit_config_t::loop_count` to `-1` means an infinite loop transmission. In this situation, the channel won’t stop until manually call of `rmt_disable()`. And the trans done event won’t be generated as well. If `rmt_transmit_config_t::loop_count` is set to a positive number, the trans done event won’t be generated until target number of loop transmission have finished. Note that, the loop transmit feature is not supported on all ESP chips, please refer to [TRM] before you configure this option. Or you might encounter ESP_ERR_NOT_SUPPORTED error.
- `rmt_transmit_config_t::eot_level` sets the output level when the transmitter finishes working or stops working by calling `rmt_disable()`.
• Increase the \texttt{rmt\_tx\_channel\_config\_t::mem\_block\_symbols}. This approach doesn’t work if the DMA backend is also enabled.
• Customize an encoder and construct a forever loop in the encoding function. See also \textit{RMT Encoder}.

Internally, \texttt{rmt\_transmit()} will construct a transaction descriptor and send to a job queue, which will be dispatched in the ISR. So it is possible that the transaction is not started yet when \texttt{rmt\_transmit()} returns. To ensure all pending transaction to complete, user can use \texttt{rmt\_tx\_wait\_all\_done()}. 

**Multiple Channels Simultaneous Transmission** In some real-time control applications, we don’t want any time drift in between when startup multiple TX channels. For example, to make two robotic arms move simultaneously. The RMT driver can help to manage this by creating a so-called \textbf{Sync Manager}. The sync manager is represented by \texttt{rmt\_sync\_manager\_handle\_t} in the driver. The procedure of RMT sync transmission is shown as follows:

![Diagram of RMT TX Sync](image)

**Install RMT Sync Manager** To create a sync manager, user needs to tell which channels are going to be managed in the \texttt{rmt\_sync\_manager\_config\_t}:

- \texttt{rmt\_sync\_manager\_config\_t::tx\_channel\_array} points to the array of TX channels to be managed.
- \texttt{rmt\_sync\_manager\_config\_t::array\_size} sets the number of channels to be managed.

\texttt{rmt\_new\_sync\_manager()} can return a manager handle on success. This function could also fail due to various errors such as invalid arguments, etc. Specially, when the sync manager has been installed before, and there’re no hardware resources to create another manager, this function will report \texttt{ESP\_ERR\_NOT\_FOUND} error. In addition, if the sync manager is not supported by the hardware, it will report \texttt{ESP\_ERR\_NOT\_SUPPORTED} error. Please refer to [TRM] before using the sync manager feature.

**Start Transmission Simultaneously** For any managed TX channel, it won’t start the machine until all the channels in the \texttt{rmt\_sync\_manager\_config\_t::tx\_channel\_array} are called with \texttt{rmt\_transmit()}.

Before that, the channel is just put in a waiting state. Different channel usually take different time to finish the job if the transaction is different, which results in a loss of sync. So user needs to call \texttt{rmt\_sync\_reset()} to pull the channels back to the starting line again before restarting a simultaneous transmission.

Calling \texttt{rmt\_del\_sync\_manager()} can recycle the sync manager and enable the channels to initiate transactions independently afterwards.
Initiate RX Transaction  As also discussed in the Enable and Disable channel, the RX channel still doesn’t get ready to receive RMT symbols even user calls rmt_enable(). User needs to specify the basic characteristics of the incoming signals in rmt_receive_config_t:

- rmt_receive_config_t::signal_range_min_ns specifies the minimal valid pulse duration (either high or low level). A pulse whose width is smaller than this value will be treated as glitch and ignored by the hardware.
- rmt_receive_config_t::signal_range_max_ns specifies the maximum valid pulse duration (either high or low level). A pulse whose width is bigger than this value will be treated as Stop Signal, and the receiver will generate receive complete event immediately.

The RMT receiver will start the RX machine after user calls rmt_receive() with the provided configuration above. Note that, this configuration is transaction specific, which means, to start a new round of reception, user needs to sets the rmt_receive_config_t again. The receiver saves the incoming signals into its internal memory block or DMA buffer, in the format of rmt_symbol_word_t. Due to the limited size of memory block, the RMT receiver can only save short frames whose length is not longer than the memory block capacity. Long frames will be truncated by the hardware, and the driver will report an error message: hw buffer too small, received symbols truncated.

The copy destination should be provided in the buffer parameter of rmt_receive() function. If this buffer size is not sufficient, the receiver can continue to work but later incoming symbols will be dropped and report an error message: user buffer too small, received symbols truncated. Please take care of the lifecycle of the buffer parameter, user shouldn’t recycle the buffer before the receiver finished or stopped working.

The receiver will be stopped by the driver when it finishes working (i.e. received a signal whose duration is bigger than rmt_receive_config_t::signal_range_max_ns). User needs to call rmt_receive() again to restart the receiver, is necessary. User can get the received data in the rmt_rx_event_callbacks_t::on_recv_done callback. See also Register Event Callbacks for more information.
static bool example_rmt_rx_done_callback(rmt_channel_handle_t channel, const rmt_rx_done_event_data_t *edata, void *user_data) {
    BaseType_t high_task_wakeup = pdFALSE;
    QueueHandle_t receive_queue = (QueueHandle_t)user_data;
    // send the received RMT symbols to the parser task
    xQueueSendFromISR(receive_queue, edata, &high_task_wakeup);
    // return whether any task is woken up
    return high_task_wakeup == pdTRUE;
}

QueueHandle_t receive_queue = xQueueCreate(1, sizeof(rmt_rx_done_event_data_t));
rmt_rx_event_callbacks_t cbs = {
    .on_recv_done = example_rmt_rx_done_callback,
};
ESP_ERROR_CHECK(rmt_rx_register_event_callbacks(rx_channel, &cbs, receive_queue));

// the following timing requirement is based on NEC protocol
rmt_receive_config_t receive_config = {
    .signal_range_min_ns = 1250,  // the shortest duration for NEC signal is...
    .signal_range_max_ns = 12000000,  // the longest duration for NEC signal is...
};
rmt_symbol_word_t raw_symbols[64]; // 64 symbols should be sufficient for a...
// ready to receive
ESP_ERROR_CHECK(rmt_receive(rx_channel, raw_symbols, sizeof(raw_symbols), &receive_config));
// wait for RX done signal
rmt_rx_done_event_data_t rx_data;
xQueueReceive(receive_queue, &rx_data, portMAX_DELAY);
// parse the receive symbols
example_parse_nec_frame(rx_data.received_symbols, rx_data.num_symbols);

RMT Encoder

An RMT encoder is part of the RMT TX transaction, whose responsibility is to generate and write the correct RMT symbols into hardware memory (or DMA buffer) at specific time. There’re some special restrictions for an encoding function:

- An encoding function might be called for several times within a single transaction. This is because the target RMT memory block can’t accommodate all the artifacts at once. We have to use the memory in a ping-pong way, thus the encoding session is divided into multiple parts. This requires the encoder to be stateful.
- The encoding function is running in the ISR context. To speed up the encoding session, it’s high recommend to put the encoding function into IRAM. This can also avoid the cache miss during encoding.

To help get started with RMT driver faster, some commonly used encoders are provided out-of-the box. They can either work alone or chained together into a new encoder. See also Composite Pattern for the principle behind. The driver has defined the encoder interface in rmt_encoder_t, it contains the following functions:

- rmt_encoder_t::encode is the fundamental function of an encoder. This is where the encoding session happens. Please note, the rmt_encoder_t::encode function might be called for multiple times within a single transaction. The encode function should return the state of current encoding session. The supported states are listed in the rmt_encode_state_t. If the result contains RMT_ENCODING_COMPLETE, it means the current encoder has finished work. If the result contains RMT_ENCODING_MEM_FULL, we need to yield from current session, as there’s no space to save more encoding artifacts.
- rmt_encoder_t::reset should reset the encoder state back to initial. The RMT encoder is stateful, if RMT transmitter stopped manually without its corresponding encoder being reset, then the following encoding session can be wrong. This function is also called implicitly in rmt_disable().
- rmt_encoder_t::del function should free the resources allocated by the encoder.
Copy Encoder  A copy encoder is created by calling `rmt_new_copy_encoder()`. Copy encoder’s main functionality is to copy the RMT symbols from user space into the driver layer. It’s usually used to encode const data (i.e. data won’t change at runtime after initialization), for example, the leading code in the IR protocol.

A configuration structure `rmt_copy_encoder_config_t` should be provided in advance before calling `rmt_new_copy_encoder()`. Currently, this configuration is reserved for future expansion.

Bytes Encoder  A bytes encoder is created by calling `rmt_new_bytes_encoder()`. Bytes encoder’s main functionality is to convert the user space byte stream into RMT symbols dynamically. It’s usually used to encode dynamic data, for example, the address and command fields in the IR protocol.

A configuration structure `rmt_bytes_encoder_config_t` should be provided in advance before calling `rmt_new_bytes_encoder()`:

- `rmt_bytes_encoder_config_t::bit0` and `rmt_bytes_encoder_config_t::bit1` are necessary to tell the encoder how to represent bit zero and bit one in the format of `rmt_symbol_word_t`.
- `rmt_bytes_encoder_config_t::msb_first` sets the encoding order for of byte. If it is set to true, the encoder will encode the Most Significant Bit first. Otherwise, it will encode the Least Significant Bit first.

Besides the primitive encoders provided by the driver, user can implement his own encoder by chaining the existing encoders together. A common encoder chain is shown as follows:

![RMT Encoder Chain Diagram](image)

图 17: RMT Encoder Chain

Customize RMT Encoder for NEC Protocol  In this section, we will demonstrate on how to write an NEC encoder. The NEC IR protocol uses pulse distance encoding of the message bits. Each pulse burst is 562.5µs in length, logical bits are transmitted as follows. It is worth mentioning, the bytes of data bits are sent least significant bit first.

- Logical 0: a 562.5µs pulse burst followed by a 562.5µs space, with a total transmit time of 1.125ms
- Logical 1: a 562.5µs pulse burst followed by a 1.6875ms space, with a total transmit time of 2.25ms

When a key is pressed on the remote controller, the message transmitted consists of the following, in order:

![IR NEC Frame Diagram](image)

图 18: IR NEC Frame
Chapter 2. API

- 9ms leading pulse burst (also called the “AGC pulse”)
- 4.5ms space
- 8-bit address for the receiving device
- 8-bit logical inverse of the address
- 8-bit command
- 8-bit logical inverse of the command
- a final 362.5µs pulse burst to signify the end of message transmission

Then we can construct the NEC rmt_encoder_t::encode function in the same order, for example:

```c
// IR NEC scan code representation
typedef struct {
    uint16_t address;
    uint16_t command;
} ir_nec_scan_code_t;

// construct a encoder by combining primitive encoders
typedef struct {
    rmt_encoder_t base;         // the base "class", declares the standard
    rmt_encoder_t *copy_encoder; // use the copy_encoder to encode the leading
    rmt_encoder_t *bytes_encoder; // use the bytes_encoder to encode the address
    rmt_symbol_word_t nec_leading_symbol; // NEC leading code with RMT
    rmt_symbol_word_t nec_ending_symbol; // NEC ending code with RMT
    int state; // record the current encoding state (i.e. we're in which encoding phase)
} rmt_ir_nec_encoder_t;

static size_t rmt_encode_ir_nec(rmt_encoder_t *encoder, rmt_channel_handle_t channel, const void *primary_data, size_t data_size, rmt_encode_state_t *ret_state) {
    rmt_ir_nec_encoder_t *nec_encoder = __containerof(encoder, rmt_ir_nec_encoder_t, base);
    rmt_encode_state_t session_state = 0;
    rmt_encode_state_t state = 0;
    size_t encoded_symbols = 0;
    ir_nec_scan_code_t *scan_code = (ir_nec_scan_code_t *)primary_data;
    rmt_encoder_handle_t copy_encoder = nec_encoder->copy_encoder;
    rmt_encoder_handle_t bytes_encoder = nec_encoder->bytes_encoder;
    switch (nec_encoder->state) {
        case 0: // send leading code
            encoded_symbols += copy_encoder->encode(copy_encoder, channel, &nec_encoder->nec_leading_symbol, sizeof(rmt_symbol_word_t), &session_state);
            if (session_state & RMT_ENCODING_COMPLETE) {
                nec_encoder->state = 1; // we can only switch to next state when current encoder finished
            }
            if (session_state & RMT_ENCODING_MEM_FULL) {
                state |= RMT_ENCODING_MEM_FULL;
                goto out; // yield if there's no free space to put other encoding artifacts
            }
            // fall-through
        case 1: // send address
            encoded_symbols += bytes_encoder->encode(bytes_encoder, channel, &scan_code->address, sizeof(uint16_t), &session_state);
            break;
    }
}
```

(continued)
if (session_state & RMT_ENCODING_COMPLETE) {
    nec_encoder->state = 2; // we can only switch to next state when
    // current encoder finished
}
if (session_state & RMT_ENCODING_MEM_FULL) {
    state |= RMT_ENCODING_MEM_FULL;
    goto out; // yield if there's no free space to put other encoding
    // artifacts
}

    // fall-through

    // send command
    encoded_symbols += bytes_encoder->encode(bytes_encoder, channel, &scan_code->command, sizeof(uint16_t), &session_state);
    if (session_state & RMT_ENCODING_COMPLETE) {
        nec_encoder->state = 3; // we can only switch to next state when
        // current encoder finished
    }
    if (session_state & RMT_ENCODING_MEM_FULL) {
        state |= RMT_ENCODING_MEM_FULL;
        goto out; // yield if there's no free space to put other encoding
        // artifacts
    }
    // fall-through

    // send ending code
    encoded_symbols += copy_encoder->encode(copy_encoder, channel, &nec_encoder->nec_ending_symbol,
        sizeof(rmt_symbol_word_t), &session_state);
    if (session_state & RMT_ENCODING_COMPLETE) {
        nec_encoder->state = 0; // back to the initial encoding session
        state |= RMT_ENCODING_COMPLETE; // telling the caller the NEC encoding
        // has finished
    }
    if (session_state & RMT_ENCODING_MEM_FULL) {
        state |= RMT_ENCODING_MEM_FULL;
        goto out; // yield if there's no free space to put other encoding
        // artifacts
    }
}

out:
    *ret_state = state;
    return encoded_symbols;

A full sample code can be found in peripherals/rmt/ir_nec_transceiver. In the above snippet, we use a switch-case plus several goto statements to implement a state machine. With this pattern, user can construct a lot more complex IR protocols.

**Power Management**  When power management is enabled (i.e. CONFIG_PM_ENABLE is on), the system will adjust the APB frequency before going into light sleep, thus potentially changing the resolution of RMT internal counter.

However, the driver can prevent the system from changing APB frequency by acquiring a power management lock of type ESP_PM_APB_FREQ_MAX. Whenever user creates an RMT channel that has selected RMT_CLK_SRC_APB as the clock source, the driver will guarantee that the power management lock is acquired after the channel enabled by rmt_enable(). Likewise, the driver releases the lock after rmt_disable() is called for the same channel. This also reveals that the rmt_enable() and rmt_disable() should appear in pairs.

If the channel clock source is selected to others like RMT_CLK_SRC_XTAL, then the driver won’t install power management lock for it, which is more suitable for a low power application as long as the source clock can still provide sufficient resolution.
IRAM Safe  By default, the RMT interrupt will be deferred when the Cache is disabled for reasons like writing/erasing the main Flash. Thus the transaction done interrupt will not get executed in time, which is not expected in a real-time application. What’s worse, when the RMT transaction relies on ping-pong interrupt to successively encode or copy RMT symbols, such delayed response can lead to an unpredictable result.

There’s a Kconfig option CONFIG_RMT_ISR_IRAM_SAFE that will:

1. Enable the interrupt being serviced even when cache is disabled
2. Place all functions that used by the ISR into IRAM\(^2\)
3. Place driver object into DRAM (in case it’s mapped to PSRAM by accident)

This Kconfig option will allow the interrupt to run while the cache is disabled but will come at the cost of increased IRAM consumption.

Thread Safety  The factory function rmt_new_tx_channel(), rmt_new_rx_channel() and rmt_new_sync_manager() are guaranteed to be thread safe by the driver, which means, user can call them from different RTOS tasks without protection by extra locks. Other functions that take the rmt_channel_handle_t and rmt_sync_manager_handle_t as the first positional parameter, are not thread safe. which means the user should avoid calling them from multiple tasks.

Kconfig Options

- CONFIG_RMT_ISR_IRAM_SAFE controls whether the default ISR handler can work when cache is disabled, see also IRAM Safe for more information.
- CONFIG_RMT_ENABLE_DEBUG_LOG is used to enabled the debug log at the cost of increased firmware binary size.

Application Examples

- RMT based RGB LED strip customized encoder: peripherals/rmt/led_strip
- RMT IR NEC protocol encoding and decoding: peripherals/rmt/ir_nec_transceiver
- RMT transactions in queue: peripherals/rmt/musical_buzzer
- RMT based stepper motor with S-curve algorithm: peripherals/rmt/stepper_motor
- RMT infinite loop for driving DShot ESC: peripherals/rmt/dshot_esc
- RMT simulate 1-wire protocol (take DS18B20 as example): peripherals/rmt/onewire_ds18b20

API Reference

Header File

- components/driver/include/driver/rmt_tx.h

Functions

esp_err_t rmt_new_tx_channel (const rmt_tx_channel_config_t *config, rmt_channel_handle_t *ret_chan)  
Create a RMT TX channel.

参数

- config [in] TX channel configurations
- ret_chan [out] Returned generic RMT channel handle

返回

- ESP_OK: Create RMT TX channel successfully
- ESP_ERR_INVALID_ARG: Create RMT TX channel failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT TX channel failed because out of memory
- ESP_ERR_NOT_FOUND: Create RMT TX channel failed because all RMT channels are used up and no more free one

\(^2\) Callback function (e.g. rmt_tx_event_callbacks_t::on_trans_done) and the functions invoked by itself should also reside in IRAM, users need to take care of this by themselves.
Chapter 2. API

- ESP_ERR_NOT_SUPPORTED: Create RMT TX channel failed because some feature is not supported by hardware, e.g. DMA feature is not supported by hardware
- ESP_FAIL: Create RMT TX channel failed because of other error

```c
esp_err_t rmt_transmit(rmt_channel_handle_t tx_channel, rmt_encoder_handle_t encoder, const void *payload, size_t payload_bytes, const rmt_transmit_config_t *config)
```

Transmit data by RMT TX channel.

### Notes:
This function will construct a transaction descriptor and push to a queue. The transaction will not start immediately until it’s dispatched in the ISR. If there’re too many transactions pending in the queue, this function will block until the queue has free space.

### Parameters:
- `tx_channel` [in] RMT TX channel that created by `rmt_new_tx_channel()`
- `encoder` [in] RMT encoder that created by various factory APIs like `rmt_new_bytes_encoder()`
- `payload` [in] The raw data to be encoded into RMT symbols
- `payload_bytes` [in] Size of the `payload` in bytes
- `config` [in] Transmission specific configuration

### Return:
- ESP_OK: Transmit data successfully
- ESP_ERR_INVALID_ARG: Transmit data failed because of invalid argument
- ESP_ERR_INVALID_STATE: Transmit data failed because channel is not enabled
- ESP_ERR_NOT_SUPPORTED: Transmit data failed because some feature is not supported by hardware, e.g. unsupported loop count
- ESP_FAIL: Transmit data failed because of other error

```c
esp_err_t rmt_tx_wait_all_done(rmt_channel_handle_t tx_channel, int timeout_ms)
```

Wait for all pending TX transactions done.

### Notes:
This function will block forever if the pending transaction can’t be finished within a limited time (e.g. an infinite loop transaction). See also `rmt_disable()` for how to terminate a working channel.

### Parameters:
- `tx_channel` [in] RMT TX channel that created by `rmt_new_tx_channel()`

### Return:
- ESP_OK: Flush transactions successfully
- ESP_ERR_INVALID_ARG: Flush transactions failed because of invalid argument
- ESP_ERR_TIMEOUT: Flush transactions failed because of timeout
- ESP_FAIL: Flush transactions failed because of other error

```c
esp_err_t rmt_tx_register_event_callbacks(rmt_channel_handle_t tx_channel, const rmt_tx_event_callbacks_t *cbs, void *user_data)
```

Set event callbacks for RMT TX channel.

### Notes:
User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

### Notes:
When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it
should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data` should also reside in SRAM.

### 参数
- `tx_channel` - [in] RMT generic channel that created by `rmt_new_tx_channel()`
- `cbs` - [in] Group of callback functions
- `user_data` - [in] User data, which will be passed to callback functions directly

### 返回
- `ESP_OK`: Set event callbacks successfully
- `ESP_ERR_INVALID_ARG`: Set event callbacks failed because of invalid argument
- `ESP_FAIL`: Set event callbacks failed because of other error

```c
esp_err_t rmt_new_sync_manager (const rmt_sync_manager_config_t *config, rmt_sync_manager_handle_t *ret_synchro)
```
Create a synchronization manager for multiple TX channels, so that the managed channel can start transmitting at the same time.

### 备注: All the channels to be managed should be enabled by `rmt_enable()` before put them into sync manager.

### 参数
- `config` - [in] Synchronization manager configuration
- `ret_synchro` - [out] Returned synchronization manager handle

### 返回
- `ESP_OK`: Create sync manager successfully
- `ESP_ERR_INVALID_ARG`: Create sync manager failed because of invalid argument
- `ESP_ERR_NOT_SUPPORTED`: Create sync manager failed because it is not supported by hardware
- `ESP_ERR_INVALID_STATE`: Create sync manager failed because not all channels are enabled
- `ESP_ERR_NO_MEM`: Create sync manager failed because out of memory
- `ESP_ERR_NOT_FOUND`: Create sync manager failed because all sync controllers are used up and no more free one
- `ESP_FAIL`: Create sync manager failed because of other error

```c
esp_err_t rmt_del_sync_manager (rmt_sync_manager_handle_t synchro)
```
Delete synchronization manager.

### 参数
- `synchro` - [in] Synchronization manager handle returned from `rmt_new_sync_manager()`

### 返回
- `ESP_OK`: Delete the synchronization manager successfully
- `ESP_ERR_INVALID_ARG`: Delete the synchronization manager failed because of invalid argument
- `ESP_FAIL`: Delete the synchronization manager failed because of other error

```c
esp_err_t rmt_sync_reset (rmt_sync_manager_handle_t synchro)
```
Reset synchronization manager.

### 参数
- `synchro` - [in] Synchronization manager handle returned from `rmt_new_sync_manager()`

### 返回
- `ESP_OK`: Reset the synchronization manager successfully
- `ESP_ERR_INVALID_ARG`: Reset the synchronization manager failed because of invalid argument
- `ESP_FAIL`: Reset the synchronization manager failed because of other error
Chapter 2. API 参考

**Structures**

struct rmt_tx_event_callbacks_t

Group of RMT TX callbacks.

备注: The callbacks are all running under ISR environment

备注： When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.

**Public Members**

*rmt_tx_done_callback_t on_trans_done

Event callback, invoked when transmission is finished

struct rmt_tx_channel_config_t

RMT TX channel specific configuration.

**Public Members**

int gpio_num

GPIO number used by RMT TX channel. Set to -1 if unused

rmt_clock_source_t clk_src

Clock source of RMT TX channel, channels in the same group must use the same clock source

uint32_t resolution_hz

Channel clock resolution, in Hz

size_t mem_block_symbols

Size of memory block, in number of rmt_symbol_word_t, must be an even

size_t trans_queue_depth

Depth of internal transfer queue, increase this value can support more transfers pending in the background

uint32_t invert_out

Whether to invert the RMT channel signal before output to GPIO pad

uint32_t with_dma

If set, the driver will allocate an RMT channel with DMA capability

uint32_t io_loop_back

The signal output from the GPIO will be fed to the input path as well

uint32_t io_od_mode

Configure the GPIO as open-drain mode
struct rmt_tx_channel_config_t::[anonymous] flags
TX channel config flags

struct rmt_transmit_config_t
RMT transmit specific configuration.

Public Members

int loop_count
Specify the times of transmission in a loop, -1 means transmitting in an infinite loop

uint32_t eot_level
Set the output level for the "End Of Transmission"

struct rmt_transmit_config_t::[anonymous] flags
Transmit config flags

struct rmt_sync_manager_config_t
Synchronous manager configuration.

Public Members

const rmt_channel_handle_t *tx_channel_array
Array of TX channels that are about to be managed by a synchronous controller

size_t array_size
Size of the tx_channel_array

Header File

• components/driver/include/driver/rmt_rx.h

Functions

esp_err_t rmt_new_rx_channel (const rmt_rx_channel_config_t *config, rmt_channel_handle_t *ret_chan)
Create a RMT RX channel.

参数
• config [in] RX channel configurations
• ret_chan [out] Returned generic RMT channel handle

返回
• ESP_OK: Create RMT RX channel successfully
• ESP_ERR_INVALID_ARG: Create RMT RX channel failed because of invalid argument
• ESP_ERR_NO_MEM: Create RMT RX channel failed because out of memory
• ESP_ERR_NOT_FOUND: Create RMT RX channel failed because all RMT channels are used up and no more free one
• ESP_ERR_NOT_SUPPORTED: Create RMT RX channel failed because some feature is not supported by hardware, e.g. DMA feature is not supported by hardware
• ESP_FAIL: Create RMT RX channel failed because of other error
### Chapter 2. API Reference

#### esp_err_t rmt_receive(rmt_channel_handle_t rx_channel, void *buffer, size_t buffer_size, const rmt_receive_config_t *config)

Initiate a receive job for RMT RX channel.

**备注:** This function is non-blocking, it initiates a new receive job and then returns. User should check the received data from the `on_recv_done` callback that registered by `rmt_rx_register_event_callbacks()`.

**参数**
- `rx_channel` 【in】RMT RX channel that created by `rmt_new_rx_channel()`
- `buffer` 【in】The buffer to store the received RMT symbols
- `buffer_size` 【in】 size of the `buffer`, in bytes
- `config` 【in】Receive specific configurations

**返回**
- ESP_OK: Initiate receive job successfully
- ESP_ERR_INVALID_ARG: Initiate receive job failed because of invalid argument
- ESP_ERR_INVALID_STATE: Initiate receive job failed because channel is not enabled
- ESP_FAIL: Initiate receive job failed because of other error

#### esp_err_t rmt_rx_register_event_callbacks(rmt_channel_handle_t rx_channel, const rmt_rx_event_callbacks_t *cbs, void *user_data)

Set callbacks for RMT RX channel.

**备注:** User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

**备注:** When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well. The `user_data` should also reside in SRAM.

**参数**
- `rx_channel` 【in】RMT generic channel that created by `rmt_new_rx_channel()`
- `cbs` 【in】Group of callback functions
- `user_data` 【in】User data, which will be passed to callback functions directly

**返回**
- ESP_OK: Set event callbacks successfully
- ESP_ERR_INVALID_ARG: Set event callbacks failed because of invalid argument
- ESP_FAIL: Set event callbacks failed because of other error

### Structures

**struct rmt_rx_event_callbacks_t**

Group of RMT RX callbacks.

**备注:** The callbacks are all running under ISR environment

**备注:** When CONFIG_RMT_ISR_IRAM_SAFE is enabled, the callback itself and functions called by it should be placed in IRAM. The variables used in the function should be in the SRAM as well.
Chapter 2. API

Public Members

\texttt{rmt\_rx\_done\_callback\_t} \texttt{on\_recv\_done}

Event callback, invoked when one RMT channel receiving transaction completes

\texttt{struct rmt\_rx\_channel\_config\_t}

RMT RX channel specific configuration.

Public Members

\texttt{int gpio\_num}

GPIO number used by RMT RX channel. Set to -1 if unused

\texttt{rmt\_clock\_source\_t} \texttt{clk\_src}

Clock source of RMT RX channel, channels in the same group must use the same clock source

\texttt{uint32\_t resolution\_hz}

Channel clock resolution, in Hz

\texttt{size\_t mem\_block\_symbols}

Size of memory block, in number of \texttt{rmt\_symbol\_word\_t}, must be an even

\texttt{uint32\_t invert\_in}

Whether to invert the incoming RMT channel signal

\texttt{uint32\_t with\_dma}

If set, the driver will allocate an RMT channel with DMA capability

\texttt{uint32\_t io\_loop\_back}

For debug/test, the signal output from the GPIO will be fed to the input path as well

\texttt{struct rmt\_rx\_channel\_config\_t::[anonymous] flags}

RX channel config flags

\texttt{struct rmt\_receive\_config\_t}

RMT receive specific configuration.

Public Members

\texttt{uint32\_t signal\_range\_min\_ns}

A pulse whose width is smaller than this threshold will be treated as glitch and ignored

\texttt{uint32\_t signal\_range\_max\_ns}

RMT will stop receiving if one symbol level has kept more than \texttt{signal\_range\_max\_ns}

Header File

- \texttt{components/driver/include/driver/rmt\_common.h}
Functions

### esp_err_t rmt_del_channel(rmt_channel_handle_t channel)
Delete an RMT channel.

- **channel** - [in] RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`

- **返回**
  - ESP_OK: Delete RMT channel successfully
  - ESP_ERR_INVALID_ARG: Delete RMT channel failed because of invalid argument
  - ESP_ERR_INVALID_STATE: Delete RMT channel failed because it is still in working
  - ESP_FAIL: Delete RMT channel failed because of other error

### esp_err_t rmt_apply_carrier(rmt_channel_handle_t channel, const rmt_carrier_config_t *config)
Apply modulation feature for TX channel or demodulation feature for RX channel.

- **channel** - [in] RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`
- **config** - [in] Carrier configuration. Specially, a NULL config means to disable the carrier modulation or demodulation feature

- **返回**
  - ESP_OK: Apply carrier configuration successfully
  - ESP_ERR_INVALID_ARG: Apply carrier configuration failed because of invalid argument
  - ESP_FAIL: Apply carrier configuration failed because of other error

### esp_err_t rmt_enable(rmt_channel_handle_t channel)
Enable the RMT channel.

- **channel** - [in] RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`

- **返回**
  - ESP_OK: Enable RMT channel successfully
  - ESP_ERR_INVALID_ARG: Enable RMT channel failed because of invalid argument
  - ESP_ERR_INVALID_STATE: Enable RMT channel failed because it’s enabled already
  - ESP_FAIL: Enable RMT channel failed because of other error

### esp_err_t rmt_disable(rmt_channel_handle_t channel)
Disable the RMT channel.

- **channel** - [in] RMT generic channel that created by `rmt_new_tx_channel()` or `rmt_new_rx_channel()`

- **返回**
  - ESP_OK: Disable RMT channel successfully
  - ESP_ERR_INVALID_ARG: Disable RMT channel failed because of invalid argument
  - ESP_ERR_INVALID_STATE: Disable RMT channel failed because it’s not enabled yet
  - ESP_FAIL: Disable RMT channel failed because of other error

Structures

struct rmt_carrier_config_t
RMT carrier wave configuration (for either modulation or demodulation)
Chapter 2. API 参考

Public Members

```c
uint32_t frequency_hz
Carrier wave frequency, in Hz, 0 means disabling the carrier
```

```c
float duty_cycle
Carrier wave duty cycle (0~100%)
```

```c
uint32_t polarity_active_low
Specify the polarity of carrier, by default it’s modulated to base signal’s high level
```

```c
uint32_t always_on
If set, the carrier can always exist even there’s not transfer undergoing
```

```c
struct rmt_carrier_config_t::[anonymous] flags
Carrier config flags
```

Header File

- components/driver/include/driver/rmt_encoder.h

Functions

```c
esp_err_t rmt_new_bytes_encoder (const rmt_bytes_encoder_config_t *config, rmt_encoder_handle_t *ret_encoder)
```

Create RMT bytes encoder, which can encode byte stream into RMT symbols.

**参数**
- `config` - [in] Bytes encoder configuration
- `ret_encoder` - [out] Returned encoder handle

**返回**
- ESP_OK: Create RMT bytes encoder successfully
- ESP_ERR_INVALID_ARG: Create RMT bytes encoder failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT bytes encoder failed because out of memory
- ESP_FAIL: Create RMT bytes encoder failed because of other error

```c
esp_err_t rmt_new_copy_encoder (const rmt_copy_encoder_config_t *config, rmt_encoder_handle_t *ret_encoder)
```

Create RMT copy encoder, which copies the given RMT symbols into RMT memory.

**参数**
- `config` - [in] Copy encoder configuration
- `ret_encoder` - [out] Returned encoder handle

**返回**
- ESP_OK: Create RMT copy encoder successfully
- ESP_ERR_INVALID_ARG: Create RMT copy encoder failed because of invalid argument
- ESP_ERR_NO_MEM: Create RMT copy encoder failed because out of memory
- ESP_FAIL: Create RMT copy encoder failed because of other error

```c
esp_err_t rmt_del_encoder (rmt_encoder_handle_t encoder)
```

Delete RMT encoder.

**参数**
- `encoder` - [in] RMT encoder handle, created by e.g rmt_new_bytes_encoder()

**返回**
- ESP_OK: Delete RMT encoder successfully
• ESP_ERR_INVALID_ARG: Delete RMT encoder failed because of invalid argument
• ESP_FAIL: Delete RMT encoder failed because of other error

`esp_err_t rmt_encoder_reset (rmt_encoder_handle_t encoder)`
Reset RMT encoder.

参数 encoder — [in] RMT encoder handle, created by `e.g. rmt_new_bytes_encoder()`
返回
• ESP_OK: Reset RMT encoder successfully
• ESP_ERR_INVALID_ARG: Reset RMT encoder failed because of invalid argument
• ESP_FAIL: Reset RMT encoder failed because of other error

**Structures**

`struct rmt_encoder_t`
Interface of RMT encoder.

**Public Members**

`size_t(*encode)(rmt_encoder_t *encoder, rmt_channel_handle_t tx_channel, const void *primary_data, size_t data_size, rmt_encode_state_t *ret_state)`
Encode the user data into RMT symbols and write into RMT memory.

备注: The encoding function will also be called from an ISR context, thus the function must not call any blocking API.

备注: It’s recommended to put this function implementation in the IRAM, to achieve a high performance and less interrupt latency.

Param encoder [in] Encoder handle
Param tx_channel [in] RMT TX channel handle, returned from `rmt_new_tx_channel()`
Param primary_data [in] App data to be encoded into RMT symbols
Param data_size [in] Size of primary_data, in bytes
Param ret_state [out] Returned current encoder’s state
Return Number of RMT symbols that the primary data has been encoded into

`esp_err_t (*reset)(rmt_encoder_t *encoder)`
Reset encoding state.

Param encoder [in] Encoder handle
Return
• ESP_OK: reset encoder successfully
• ESP_FAIL: reset encoder failed

`esp_err_t (*del)(rmt_encoder_t *encoder)`
Delete encoder object.

Param encoder [in] Encoder handle
Return
• ESP_OK: delete encoder successfully
• ESP_FAIL: delete encoder failed
struct rmt_bytes_encoder_config_t
Bytes encoder configuration.

Public Members

rmt_symbol_word_t bit0
How to represent BIT0 in RMT symbol

rmt_symbol_word_t bit1
How to represent BIT1 in RMT symbol

uint32_t msb_first
Whether to encode MSB bit first

struct rmt_bytes_encoder_config_t::[anonymous] flags
Encoder config flag

struct rmt_copy_encoder_config_t
Copy encoder configuration.

Type Definitions
typedef struct rmt_encoder_t rmt_encoder_t
Type of RMT encoder.

Enumerations
enum rmt_encode_state_t
RMT encoding state.

Values:

enumerator RMT_ENCODING_COMPLETE
The encoding session is finished, the caller can continue with subsequent encoding

enumerator RMT_ENCODING_MEM_FULL
The encoding artifact memory is full, the caller should return from current encoding session

Header File

• components/driver/include/driver/rmt_types.h

Structures

struct rmt_tx_done_event_data_t
Type of RMT TX done event data.
Public Members

size_t num_symbols
The number of transmitted RMT symbols (only one round is counted if it’s a loop transmission)

struct rmt_rx_done_event_data_t
Type of RMT RX done event data.

Public Members

rmt_symbol_word_t *received_symbols
Point to the received RMT symbols

size_t num_symbols
The number of received RMT symbols

Type Definitions

typedef struct rmt_channel_t *rmt_channel_handle_t
Type of RMT channel handle.

typedef struct rmt_sync_manager_t *rmt_sync_manager_handle_t
Type of RMT synchronization manager handle.

typedef struct rmt_encoder_t *rmt_encoder_handle_t
Type of RMT encoder handle.

typedef bool (*rmt_tx_done_callback_t)(rmt_channel_handle_t tx_chan, const rmt_tx_done_event_data_t *edata, void *user_ctx)
Prototype of RMT event callback.

Param tx_chan [in] RMT channel handle, created from rmt_new_tx_channel()
Param edata [in] Point to RMT event data. The lifecycle of this pointer memory is inside this function, user should copy it into static memory if used outside this function.
Param user_ctx [in] User registered context, passed from rmt_tx_register_event_callbacks()
Return Whether a high priority task has been waken up by this callback function

typedef bool (*rmt_rx_done_callback_t)(rmt_channel_handle_t rx_chan, const rmt_rx_done_event_data_t *edata, void *user_ctx)
Prototype of RMT event callback.

Param rx_chan [in] RMT channel handle, created from rmt_new_rx_channel()
Param edata [in] Point to RMT event data. The lifecycle of this pointer memory is inside this function, user should copy it into static memory if used outside this function.
Param user_ctx [in] User registered context, passed from rmt_rx_register_event_callbacks()
Return Whether a high priority task has been waken up by this function

Header File

- components/hal/include/hal/rmt_types.h
Unions

union rmt_symbol_word_t
#include <rmt_types.h> The layout of RMT symbol stored in memory, which is decided by the hardware design.

Public Members

unsigned int duration0
Duration of level0

unsigned int level0
Level of the first part

unsigned int duration1
Duration of level1

unsigned int level1
Level of the second part

struct rmt_symbol_word_t::[anonymous] [anonymous]

unsigned int val
Equivalent unsigned value for the RMT symbol

Type Definitions

typedef soc_periph_rmt_clk_src_t rmt_clock_source_t
RMT group clock source.

**2.6.15 SD Pull-up Requirements**

Espressif hardware products are designed for multiple use cases which may require different pull states on pins. For this reason, the pull state of particular pins on certain products will need to be adjusted to provide the pull-ups required in the SD bus.

SD pull-up requirements apply to cases where ESP32 uses the SPI or SDMMC controller to communicate with SD cards. When an SD card is operating in SPI mode or 1-bit SD mode, the CMD and DATA (DAT0 - DAT3) lines of the SD bus must be pulled up by 10 kOhm resistors. SD cards and SDIO devices should also have pull-ups on all above-mentioned lines (regardless of whether these lines are connected to the host) in order to prevent them from entering a wrong state.

By default, the MTDI bootstrapping pin is incompatible with the DAT2 line pull-up if the flash voltage is 3.3 V. For more information, see MTDI Strapping Pin below.

This document has the following structure:

- **Overview of compatibility** between the default pull states on pins of Espressif’s products and the states required by the SD bus
- **Solutions** - ideas on how to resolve compatibility issues
- **Related information** - other relevant information
Overview of Compatibility

This section provides an overview of compatibility issues that might occur when using SDIO (secure digital input output). Since the SD bus needs to be connected to pull-ups, these issues should be resolved regardless of whether they are related to master (host) or slave (device). Each issue has links to its respective solution. A solution for a host and device may differ.

Systems on a Chip (SoCs)

- ESP32 (except for D2WD versions, see ESP32 datasheet):
  - No Pull-ups
  - Conflicts Between Bootstrap and SDIO on DAT2 for models with 3.3 V flash chip
- ESP32-D2WD:
  - No Pull-ups
  - No Pull-up on GPIO12

Systems in Packages (SIP)

- ESP32-PICO-D4:
  - No Pull-ups
  - Conflicts Between Bootstrap and SDIO on DAT2

Modules

  - No Pull-ups
  - Conflicts Between Bootstrap and SDIO on DAT2
- ESP32-WROVER Series, including ESP32-WROVER and ESP32-WROVER-I
  - No Pull-ups
- ESP32-WROVER-B Series, including ESP32-WROVER-B and ESP32-WROVER-IB
  - No Pull-ups
  - Conflicts Between Bootstrap and SDIO on DAT2

Development Boards

- ESP32-PICO-KIT, including PICO-KIT v4.1, v4.0, and v3
  - No Pull-ups
  - Conflicts Between Bootstrap and SDIO on DAT2
  - Download Mode Not Working (minor issue)
- ESP32-DevKitC, including ESP32-DevKitC v4 and v2
  - No Pull-ups
  - Conflicts Between Bootstrap and SDIO on DAT2
  - Download Mode Not Working (minor issue)
- ESP-WROVER-KIT
  - Required pull-ups are provided
  - Pull-up Conflicts on GPIO13 (v4.1, v3, v2, and v1)
  - Conflicts Between Bootstrap and SDIO on DAT2 (v4.1, v2, and v1)
  - Download Mode Not Working (minor issue) (v2, v1)
  - You can determine the version of your ESP32-WROVER-KIT by checking which module is mounted on it:
    - ESP32-WROVER-B on v4.1
    - ESP32-WROVER on v3
    - ESP32-WROOM-32 on v1 and v2
- ESP32-LyraTD-MSC
  - Required pull-ups are provided
  - Conflicts Between Bootstrap and SDIO on DAT2
• ESP32-LyraT
  – Required pull-ups are provided
  – Pull-up Conflicts on GPIO13

Non-Espressif Hosts Please make sure that your SDIO host provides necessary pull-ups for all SD bus signals.

Solutions

No Pull-ups If you use a development board without pull-ups, you can do the following:

• If your host and slave device are on separate boards, replace one of them with a board that has pull-ups. For the list of Espressif’s development boards with pull-ups, go to Development Boards.
• Attach external pull-ups by connecting each pin which requires a pull-up to VDD via a 10 kOhm resistor.

Pull-up Conflicts on GPIO13 If DAT3 of your device is not properly pulled up, you have the following options:

• Use 1-bit SD mode and tie the device’s DAT3 to VDD
• Use SPI mode
• Perform one of the following actions on the GPIO13 pin:
  – Remove the pull-down resistors
  – Attach a pull-up resistor of less than 5 kOhm (2 kOhm suggested)
  – Pull it up or drive it high either by using the host or with 3.3 V on VDD in 1-bit SD mode

Conflicts Between Bootstrap and SDIO on DAT2 There is a conflict between the boot strapping requirements of the ESP32 and the SDIO protocol. For details, see MTDI Strapping Pin.

To resolve the conflict, you have the following options:

1. (Recommended) Burn the flash voltage selection eFuses. This will permanently configure the internal regulator’s output voltage to 3.3 V, and GPIO12 will not be used as a bootstrapping pin. After that, connect a pull-up resistor to GPIO12.

警告： Burning eFuses is irreversible! The issue list above might be out of date, so please make sure that the module you are burning has a 3.3 V flash chip by checking the information on https://www.espressif.com/. If you burn the 3.3 V eFuses on a module with a 1.8 V flash chip, the module will stop functioning.

If you are sure that you need to irreversibly burn eFuses, go to your ESP-IDF directory and run the following command:

```
components/esptool_py/esptool/espefuse.py set_flash_voltage 3.3V
```

This command will burn the XPD_SDIO_TIEH, XPD_SDIO_FORCE, and XPD_SDIO_REG eFuses. After all the three eFuses are burned to value 1, the internal VDD_SDIO flash voltage regulator will be permanently set to 3.3 V. You will see the following log if the burning succeeds:

```
espefuse.py v2.6
Connecting....

Enable internal flash voltage regulator (VDD_SDIO) to 3.3 V.
The following eFuses are burned: XPD_SDIO_FORCE, XPD_SDIO_REG, XPD_SDIO_TIEH.
This is an irreversible operation.
Type 'BURN' (all capitals) to continue.
BURN
VDD_SDIO setting complete.
```

To check the status of the eFuses, run:
If running from an automated flashing script, espefuse.py has an option `--do-not-confirm`. For more details, see ESP32 Technical Reference Manual [PDF].

2. **If using 1-bit SD mode or SPI mode**, disconnect the DAT2 pin and make sure it is pulled high. For this, do one of the following:
   - Leave the host’s DAT2 floating and directly connect the slave’s DAT2 to VDD.
   - For a slave device, build a firmware with the option `SDIO_SLAVE_FLAG_DAT2_DISABLED` and reflash your device. This option will help avoid slave detecting on the DAT2 line. Note that 4-bit SD mode will no longer be supported by the standard Card Common Control Register (CCCR); however, the host will not be aware of that. The use of 4-bit SD mode will have to be disabled on the host’s side.

**No Pull-up on GPIO12** Your module is compatible with the SDIO protocol. Just connect GPIO12 to VDD via a 10 kOhm resistor.

**Download Mode Not Working (minor issue)** When the GPIO2 pin is pulled high in accordance with the SD pull-up requirements, you cannot enter Download mode because GPIO2 is a bootstrapping pin which in this case must be pulled low.

There are the following solutions:

- For boards that require shorting the GPIO0 and GPIO2 pins with a jumper, put the jumper in place, and the auto-reset circuit will pull GPIO2 low along with GPIO0 before entering Download mode.
- For boards with components attached to their GPIO2 pin (such as pull-down resistors and/or LEDs), check the schematic of your development board for anything connected to GPIO2.
  - **LEDs** would not affect operation in most cases.
  - **Pull-down resistors** can interfere with DAT0 signals and must be removed.

If the above solutions do not work for you, please determine if it is the host or slave device that has pull-ups affecting their GPIO2, then locate these pull-ups and remove them.

**Related Information**

**MTDI Strapping Pin** MTDI (GPIO12) is used as a bootstrapping pin to select the output voltage of an internal regulator (VDD_SDIO) which powers the flash chip. This pin has an internal pull-down, so, if left unconnected, it will read low level at startup, which will lead to selecting the default 3.3 V operation.

All ESP32-WROVER modules, excluding ESP32-WROVER-B, use 1.8 V flash and have internal pull-ups on GPIO12. Other modules that use 3.3 V flash have no pull-ups on the GPIO12 pin, and this pin is slightly pulled down internally.

When adding a pull-up to this pin for SD card operation, consider the following:

- For boards that do not use the internal regulator (VDD_SDIO) to power flash, GPIO12 can be pulled high.
- For boards using 1.8 V flash chips, GPIO12 needs to be pulled high at reset. This is fully compatible with the SD card operation.
- On boards using the internal regulator and a 3.3 V flash chip, GPIO12 must be pulled low at reset. This is incompatible with the SD card operation. For reference information on compatibility of Espressif’s boards with the SD card operation, see **Overview of Compatibility**.

**Internal Pull-ups and Strapping Requirements** Using external resistors is always preferable. However, Espressif’s products have internal weak pull-up and pull-down resistors which can be enabled and used instead of external ones. Please keep in mind that this solution cannot guarantee reliable SDIO communication.

With that said, the information about these internal pull-ups and strapping requirements can still be useful. Espressif hardware products have different weak internal pull-ups/pull-downs connected to CMD and DATA pins. The table below shows the default pull-up and pull-down states of the CMD and DATA pins.
The following abbreviations are used in the table:

- **WPU**: Weak pull-up inside the SoC
- **WPD**: Weak pull-down inside the SoC
- **PU**: Pull-up inside Espressif modules but outside the SoC

### Table 6: Default pull-up and pull-down states of the CMD and DATA pins

<table>
<thead>
<tr>
<th>GPIO number</th>
<th>Pin Name</th>
<th>Startup State</th>
<th>Strapping Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>CMD</td>
<td>WPU</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DAT0</td>
<td>WPD</td>
<td>Low for Download mode</td>
</tr>
<tr>
<td>4</td>
<td>DAT1</td>
<td>WPD</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>DAT2</td>
<td>PU for 1.8 V flash; WPD for 3.3 V flash</td>
<td>High for 1.8 V flash; Low for 3.3 V flash</td>
</tr>
<tr>
<td>13</td>
<td>DAT3</td>
<td>WPU</td>
<td></td>
</tr>
</tbody>
</table>

### 2.6.16 SDMMC Host Driver

**Overview**

ESP32’s SDMMC host peripheral has two slots. Each slot can be used independently to connect to an SD card, SDIO device or eMMC chip.

- Slot 0 (SDMMC_HOST_SLOT_0) is an 8-bit slot. It uses HS1_* signals in the PIN MUX.
- Slot 1 (SDMMC_HOST_SLOT_1) is a 4-bit slot. It uses HS2_* signals in the PIN MUX.

The slots are connected to ESP32 GPIOs using IO MUX. Pin mappings of these slots are given in the table below.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Slot 0</th>
<th>Slot 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD</td>
<td>GPIO11</td>
<td>GPIO15</td>
</tr>
<tr>
<td>CLK</td>
<td>GPIO06</td>
<td>GPIO14</td>
</tr>
<tr>
<td>D0</td>
<td>GPIO07</td>
<td>GPIO02</td>
</tr>
<tr>
<td>D1</td>
<td>GPIO08</td>
<td>GPIO04</td>
</tr>
<tr>
<td>D2</td>
<td>GPIO09</td>
<td>GPIO12</td>
</tr>
<tr>
<td>D3</td>
<td>GPIO10</td>
<td>GPIO13</td>
</tr>
<tr>
<td>D4</td>
<td>GPIO16</td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>GPIO17</td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>GPIO05</td>
<td></td>
</tr>
<tr>
<td>D7</td>
<td>GPIO18</td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>any input via GPIO matrix</td>
<td></td>
</tr>
<tr>
<td>WP</td>
<td>any input via GPIO matrix</td>
<td></td>
</tr>
</tbody>
</table>

The Card Detect and Write Protect signals can be routed to arbitrary pins using the GPIO matrix. To reserve the pins, set the cd and wp members of the sdmmc_slot_config_t structure before calling sdmmc_host_init_slot(). Please note that it is not advised to specify a Card Detect pin when working with SDIO cards, because the card detect signal in ESP32 can also trigger SDIO slave interrupt.

**WARNING**: Pins used by Slot 0 (HS1_* ) are also used to connect the SPI flash chip in ESP32-WROOM and ESP32-WROVER modules. These pins cannot be shared between an SD card and SPI flash. If you need to use Slot 0, connect SPI flash to different pins and set eFuses accordingly.

**Supported Speed Modes**

SDMMC Host driver supports the following speed modes:

- Default Speed (20 MHz), 1/4-line (with SD cards), and 1/4/8-line (with 3.3 V eMMC)
Chapter 2. API

- High Speed (40 MHz), 1/4-line (with SD cards), and 1/4/8-line (with 3.3 V eMMC)
- High Speed DDR (40 MHz), 4-line (with 3.3 V eMMC)

Speed modes not supported at present:
- High Speed DDR mode, 8-line eMMC
- UHS-I 1.8 V modes, 4-line SD cards

Using the SDMMC Host Driver

Of all the functions listed below, only the following ones will be used directly by most applications:

- `sdmmc_host_init()`
- `sdmmc_host_init_slot()`
- `sdmmc_host_deinit()`

Other functions, such as the ones given below, will be called by the SD/MMC protocol layer via function pointers in the `sdmmc_host_t` structure:

- `sdmmc_host_set_bus_width()`
- `sdmmc_host_set_card_clk()`
- `sdmmc_host_do_transaction()`

Configuring Bus Width and Frequency

With the default initializers for `sdmmc_host_t` and `sdmmc_slot_config_t` (`SDMMC_HOST_DEFAULT` and `SDMMC_SLOT_CONFIG_DEFAULT`), SDMMC Host driver will attempt to use the widest bus supported by the card (4 lines for SD, 8 lines for eMMC) and the frequency of 20 MHz.

In the designs where communication at 40 MHz frequency can be achieved, it is possible to increase the bus frequency by changing the `max_freq_khz` field of `sdmmc_host_t`:

```
sdmmc_host_t host = SDMMC_HOST_DEFAULT();
host.max_freq_khz = SDMMC_FREQ_HIGHSPEED;
```

If you need a specific frequency other than standard speeds, you are free to use any value from within appropriate range of the SD interface given (SDMMC or SDSPI). However, the real clock frequency shall be calculated by the underlying driver and the value can be different from the one required. For the SDMMC, `max_freq_khz` works as the upper limit so the final frequency value shall be always lower or equal. For the SDSPI, the nearest fitting frequency is supplied and thus the value can be greater than / equal to / lower than `max_freq_khz`.

To configure the bus width, set the `width` field of `sdmmc_slot_config_t`. For example, to set 1-line mode:

```
sdmmc_slot_config_t slot = SDMMC_SLOT_CONFIG_DEFAULT();
slot.width = 1;
```

DDR Mode for eMMC chips

By default, DDR mode will be used if:

- SDMMC host frequency is set to `SDMMC_FREQ_HIGHSPEED` in `sdmmc_host_t` structure, and
- eMMC chip reports DDR mode support in its CSD register

DDR mode places higher requirements for signal integrity. To disable DDR mode while keeping `SDMMC_FREQ_HIGHSPEED` frequency, clear `SDMMC_HOST_FLAG_DDR` bit in flags field of `sdmmc_host_t`:

```
sdmmc_host_t host = SDMMC_HOST_DEFAULT();
host.max_freq_khz = SDMMC_FREQ_HIGHSPEED;
host.flags &= ~SDMMC_HOST_FLAG_DDR;
```
See also

See **SD/SDIO/MMC Driver** for the higher level driver which implements the protocol layer.

See **SD SPI Host Driver** for a similar driver which uses the SPI controller and is limited to SD protocol’s SPI mode.

See **SD Pull-up Requirements** for pullup support and compatibilities of modules and development kits.

### API Reference

#### Header File

- components/driver/include/driver/sdmmc_host.h

#### Functions

**esp_err_t sdmmc_host_init**(void)

Initialize SDMMC host peripheral.

备注：This function is not thread safe

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init was already called
- ESP_ERR_NO_MEM if memory cannot be allocated

**esp_err_t sdmmc_host_init_slot**(int slot, const sdmmc_slot_config_t *slot_config)

Initialize given slot of SDMMC peripheral.

On the ESP32, SDMMC peripheral has two slots:

- Slot 0: 8-bit wide, maps to HS1_* signals in PIN MUX
- Slot 1: 4-bit wide, maps to HS2_* signals in PIN MUX

Card detect and write protect signals can be routed to arbitrary GPIOs using GPIO matrix.

备注：This function is not thread safe

**参数

- slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- slot_config - additional configuration for the slot

**返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if host has not been initialized using sdmmc_host_init

**esp_err_t sdmmc_host_set_bus_width**(int slot, size_t width)

Select bus width to be used for data transfer.

SD/MMC card must be initialized prior to this command, and a command to set bus width has to be sent to the card (e.g. SD_APP_SET_BUS_WIDTH)

备注：This function is not thread safe

**参数

- slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- width - bus width (1, 4, or 8 for slot 0; 1 or 4 for slot 1)
Chapter 2. API Reference

size_t sdmmc_host_get_slot_width (int slot)
Get bus width configured in sdmmc_host_init_slot to be used for data transfer.

参数 slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)

返回 ESP_OK on success
ESP_ERR_INVALID_ARG if slot number or width is not valid

esp_err_t sdmmc_host_set_card_clk (int slot, uint32_t freq_khz)
Set card clock frequency.
Currently only integer fractions of 40MHz clock can be used. For High Speed cards, 40MHz can be used. For Default Speed cards, 20MHz can be used.

备注: This function is not thread safe

参数
slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
freq_khz - card clock frequency, in kHz

返回 ESP_OK on success
other error codes may be returned in the future

esp_err_t sdmmc_host_set_bus_ddr_mode (int slot, bool ddr_enabled)
Enable or disable DDR mode of SD interface.

参数
slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
ddr_enabled - enable or disable DDR mode

返回 ESP_OK on success
ESP_ERR_NOT_SUPPORTED if DDR mode is not supported on this slot

esp_err_t sdmmc_host_do_transaction (int slot, sdmmc_command_t *cmdinfo)
Send command to the card and get response.
This function returns when command is sent and response is received, or data is transferred, or timeout occurs.

Attention Data buffer passed in cmdinfo->data must be in DMA capable memory

备注: This function is not thread safe w.r.t. init/deinit functions, and bus width/clock speed configuration functions. Multiple tasks can call sdmmc_host_do_transaction as long as other sdmmc_host_* functions are not called.

参数
slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
cmdinfo - pointer to structure describing command and data to transfer

返回 ESP_OK on success
ESP_ERR_TIMEOUT if response or data transfer has timed out
ESP_ERR_INVALID_CRC if response or data transfer CRC check has failed
ESP_ERR_INVALID_RESPONSE if the card has sent an invalid response
ESP_ERR_INVALID_SIZE if the size of data transfer is not valid in SD protocol
ESP_ERR_INVALID_ARG if the data buffer is not in DMA capable memory
### Chapter 2. API 参考

**esp_err_t sdmmc_host_io_int_enable** (int slot)

Enable IO interrupts.

This function configures the host to accept SDIO interrupts.

参数: slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)

返回: returns ESP_OK, other errors possible in the future

**esp_err_t sdmmc_host_io_int_wait** (int slot, TickType_t timeout_ticks)

Block until an SDIO interrupt is received, or timeout occurs.

参数:
- slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- timeout_ticks - number of RTOS ticks to wait for the interrupt

返回:
- ESP_OK on success (interrupt received)
- ESP_ERR_TIMEOUT if the interrupt did not occur within timeout_ticks

**esp_err_t sdmmc_host_deinit** (void)

Disable SDMMC host and release allocated resources.

备注: This function is not thread safe

返回:
- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init function has not been called

**esp_err_t sdmmc_host_get_real_freq** (int slot, int *real_freq_khz)

Provides a real frequency used for an SD card installed on specific slot of SD/MMC host controller.

This function calculates real working frequency given by current SD/MMC host controller setup for required slot: it reads associated host and card dividers from corresponding SDMMC registers, calculates respective frequency and stores the value into the ‘real_freq_khz’ parameter

参数:
- slot - slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- real_freq_khz - [out] output parameter for the result frequency (in kHz)

返回:
- ESP_OK on success
- ESP_ERR_INVALID_ARG on real_freq_khz == NULL or invalid slot number used

### Structures

**struct sdmmc_slot_config_t**

Extra configuration for SDMMC peripheral slot

### Public Members

**gpio_num_t gpio_cd**

GPIO number of card detect signal.

**gpio_num_t cd**

GPIO number of card detect signal; shorter name.
**Chapter 2. API**

```c
gpio_num_t gpio_wp
GPIO number of write protect signal.
```

```c
gpio_num_t wp
GPIO number of write protect signal; shorter name.
```

```c
uint8_t width
Bus width used by the slot (might be less than the max width supported)
```

```c
uint32_t flags
Features used by this slot.
```

**Macros**

```c
SDMMC_HOST_SLOT_0
SDMMC slot 0.
```

```c
SDMMC_HOST_SLOT_1
SDMMC slot 1.
```

```c
SDMMC_HOST_DEFAULT()
Default sdmmc_host_t structure initializer for SDMMC peripheral.
Uses SDMMC peripheral, with 4-bit mode enabled, and max frequency set to 20MHz
```

```c
SDMMC_SLOT_FLAG_INTERNAL_PULLUP
Enable internal pullups on enabled pins. The internal pullups are insufficient however, please make sure external pullups are connected on the bus. This is for debug / example purpose only.
```

```c
SDMMC_SLOT_NO_CD
indicates that card detect line is not used
```

```c
SDMMC_SLOT_NO_WP
indicates that write protect line is not used
```

```c
SDMMC_SLOT_WIDTH_DEFAULT
use the maximum possible width for the slot
```

```c
SDMMC_SLOT_CONFIG_DEFAULT()
Macro defining default configuration of SDMMC host slot
```

**2.6.17 SD SPI Host Driver**

**Overview**

The SD SPI host driver allows communicating with one or more SD cards by the SPI Master driver which makes use of the SPI host. Each card is accessed through an SD SPI device represented by an `sdspi_dev_handle_t` returned when attaching the device to an SPI bus by calling `sdspi_host_init_device`. The bus should be already initialized before (by `spi_bus_initialize`).

This driver’s naming pattern was adopted from the SDMMC Host driver due to their similarity. Likewise, the APIs of both drivers are also very similar.
SD SPI driver (access the SD card in SPI mode) offers lower throughput but makes pin selection more flexible. With the help of the GPIO matrix, an SPI peripheral’s signals can be routed to any ESP32 pin. Otherwise, if SDMMC host driver is used (see SDMMC Host) to access the card in SD 1-bit/4-bit mode, higher throughput can be reached but it requires routing the signals through their dedicated IO_MUX pins only.

With the help of SPI Master driver based on, the SPI bus can be shared among SD cards and other SPI devices. The SPI Master driver will handle exclusive access from different tasks.

The SD SPI driver uses software-controlled CS signal.

**How to Use**

Firstly, use the macro `SDSPI_DEVICE_CONFIG_DEFAULT` to initialize a structure `sdspi_device_config_t`, which is used to initialize an SD SPI device. This macro will also fill in the default pin mappings, which is same as the pin mappings of SDMMC host driver. Modify the host and pins of the structure to desired value. Then call `sdspi_host_init_device` to initialize the SD SPI device and attach to its bus.

Then use `SDSPI_HOST_DEFAULT` macro to initialize a `sdmmc_host_t` structure, which is used to store the state and configurations of upper layer (SD/SDIO/MMC driver). Modify the `slot` parameter of the structure to the SD SPI device `spi_handle` just returned from `sdspi_host_init_device`. Call `sdmmc_card_init` with the `sdmmc_host_t` to probe and initialize the SD card.

Now you can use SD/SDIO/MMC driver functions to access your card!

**Other Details**

Only the following driver’s API functions are normally used by most applications:

- `sdspi_host_init()`
- `sdspi_host_init_device()`
- `sdspi_host_remove_device()`
- `sdspi_host_deinit()`

Other functions are mostly used by the protocol level SD/SDIO/MMC driver via function pointers in the `sdmmc_host_t` structure. For more details, see the SD/SDIO/MMC Driver.

备注: SD over SPI does not support speeds above `SDMMC_FREQ_DEFAULT` due to the limitations of the SPI driver.

警告: If you want to share the SPI bus among SD card and other SPI devices, there are some restrictions, see Sharing the SPI bus among SD card and other SPI devices.

**Related Docs**

**Sharing the SPI bus among SD card and other SPI devices** The SD card has a SPI mode, which allows it to be communicated to as a SPI device. But there are some restrictions that we need to pay attention to.

**Pin loading of other devices** When adding more devices onto the same bus, the overall pin loading increases. The loading consists of AC loading (pin capacitor) and DC loading (pull-ups).

**AC loading** SD cards, which are designed for high-speed communications, have small pin capacitors (AC loading) to work until 50MHz. However, the other attached devices will increase the pin’s AC loading.

Heavy AC loading of a pin may prevent the pin from being toggled quickly. By using an oscilloscope, you will see the edges of the pin become smoother and not ideal any more (the gradient of the edge is smaller). The setup timing
requirements of an SD card may be violated when the card is connected to such bus. Even worse, the clock from the host may not be recognized by the SD card and other SPI devices on the same bus.

This issue may be more obvious if other attached devices are not designed to work at the same frequency as the SD card, because they may have larger pin capacitors.

To see if your pin AC loading is too heavy, you can try the following tests:

(Terminology: launch edge: at which clock edge the data start to toggle; latch edge: at which clock edge the data is supposed to be sampled by the receiver, for SD card, it’s the rising edge.)

1. Use an oscilloscope to see the clock and compare the data line to the clock. - If you see the clock is not fast enough (for example, the rising/falling edge is longer than 1/4 of the clock cycle), it means the clock is skewed too much. - If you see the data line unstable before the latch edge of the clock, it means the load of the data line is too large.
   You may also observed the corresponding phenomenon (data delayed largely from launching edge of clock) with logic analyzers. But it’s not as obvious as with an oscilloscope.
2. Try to use slower clock frequency.
   If the lower frequency can work while the higher frequency can’t, it’s an indication of the AC loading on the pins is too large.

If the AC loading of the pins is too large, you can either use other faster devices (with lower pin load) or slow down the clock speed.

**DC loading**

The pull-ups required by SD cards are usually around 10 kOhm to 50 kOhm, which may be too strong for some other SPI devices.

Check the specification of your device about its DC output current, it should be larger than 700uA, otherwise the device output may not be read correctly.

**Initialization sequence**

备注： If you see any problem in the following steps, please make sure the timing is correct first. You can try to slow down the clock speed (SDMMC_FREQ_PROBING = 400 KHz for SD card) to avoid the influence of pin AC loading (see above section).

When using ab SD card with other SPI devices on the same SPI bus, due to the restrictions of the SD card startup flow, the following initialization sequence should be followed: (See also storage/sd_card)

1. Initialize the SPI bus properly by `spi_bus_initialize`.
2. Tie the CS lines of all other devices than the SD card to high. This is to avoid conflicts to the SD card in the following step.
   You can do this by either:
   1. Attach devices to the SPI bus by calling `spi_bus_add_device`. This function will initialize the GPIO that is used as CS to the idle level: high.
   2. Initialize GPIO on the CS pin that needs to be tied up before actually adding a new device.
   3. Rely on the internal/external pull-up (not recommended) to pull-up all the CS pins when the GPIOs of ESP are not initialized yet. You need to check carefully the pull-up is strong enough and there are no other pull-downs that will influence the pull-up (For example, internal pull-down should be enabled).
3. Mount the card to the filesystem by calling `esp_vfs_fat_sdspi_mount`.
   This step will put the SD card into the SPI mode, which SHOULD be done before all other SPI communications on the same bus. Otherwise the card will stay in the SD mode, in which mode it may randomly respond to any SPI communications on the bus, even when its CS line is not addressed.
   If you want to test this behavior, please also note that, once the card is put into SPI mode, it will not return to SD mode before next power cycle, i.e. powered down and powered up again.
4. Now you can talk to other SPI devices freely!

**API Reference**

**Header File**
Chapter 2. API 参考

- components/driver/include/driver/sdspi_host.h

**Functions**

```c
esp_err_t sdspi_host_init (void)
```

Initialize SD SPI driver.

备注：This function is not thread safe.

返回

- ESP_OK on success
- other error codes may be returned in future versions

```c
esp_err_t sdspi_host_init_device (const sdspi_device_config_t *dev_config, sdspi_dev_handle_t *out_handle)
```

Attach and initialize an SD SPI device on the specific SPI bus.

备注：This function is not thread safe.

备注：Initialize the SPI bus by `spi_bus_initialize()` before calling this function.

备注：The SDIO over sdspi needs an extra interrupt line. Call `gpio_install_isr_service()` before this function.

参数

- `dev_config` - pointer to device configuration structure
- `out_handle` - Output of the handle to the sdspi device.

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if `sdspi_host_init_device` has invalid arguments
- ESP_ERR_NO_MEM if memory can not be allocated
- other errors from the underlying spi_master and gpio drivers

```c
esp_err_t sdspi_host_remove_device (sdspi_dev_handle_t handle)
```

Remove an SD SPI device.

参数 `handle` - Handle of the SD SPI device

返回 Always ESP_OK

```c
esp_err_t sdspi_host_do_transaction (sdspi_dev_handle_t handle, sdmmc_command_t *cmdinfo)
```

Send command to the card and get response.

This function returns when command is sent and response is received, or data is transferred, or timeout occurs.

备注：This function is not thread safe w.r.t. init/deinit functions, and bus width/clock speed configuration functions. Multiple tasks can call `sdspi_host_do_transaction` as long as other `sdspi_host_*` functions are not called.

参数

- `handle` - Handle of the sdspi device
- `cmdinfo` - pointer to structure describing command and data to transfer
Chapter 2. API

- ESP_OK on success
- ESP_ERR_TIMEOUT if response or data transfer has timed out
- ESP_ERR_INVALID_CRC if response or data transfer CRC check has failed
- ESP_ERR_INVALID_RESPONSE if the card has sent an invalid response

```c
esp_err_t sdspi_host_set_card_clk (sdspi_dev_handle_t host, uint32_t freq_khz)
```

Set card clock frequency.

Currently only integer fractions of 40MHz clock can be used. For High Speed cards, 40MHz can be used. For Default Speed cards, 20MHz can be used.

```
备注: This function is not thread safe
```

参数

- host - Handle of the sdspi device
- freq_khz - card clock frequency, in kHz

返回

- ESP_OK on success
- other error codes may be returned in the future

```c
esp_err_t sdspi_host_get_real_freq (sdspi_dev_handle_t handle, int *real_freq_khz)
```

Calculate working frequency for specific device.

参数

- handle - SDSPI device handle
- real_freq_khz - [out] output parameter to hold the calculated frequency (in kHz)

返回

- ESP_ERR_INVALID_ARG : handle is NULL or invalid or real_freq_khz parameter is NULL
- ESP_OK : Success

```c
esp_err_t sdspi_host_deinit (void)
```

Release resources allocated using sdspi_host_init.

```
备注: This function is not thread safe
```

返回

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdspi_host_init function has not been called

```c
esp_err_t sdspi_host_io_int_enable (sdspi_dev_handle_t handle)
```

Enable SDIO interrupt.

参数 handle - Handle of the sdspi device

返回 ESP_OK on success

```c
esp_err_t sdspi_host_io_int_wait (sdspi_dev_handle_t handle, TickType_t timeout_ticks)
```

Wait for SDIO interrupt until timeout.

参数 handle - Handle of the sdspi device
- timeout_ticks - Ticks to wait before timeout.

返回 ESP_OK on success
Structures

struct sdspi_device_config_t
Extra configuration for SD SPI device.

Public Members

spi_host_device_t host_id
SPI host to use, SPIx_HOST (see spi_types.h).

gpio_num_t gpio_cs
GPIO number of CS signal.

gpio_num_t gpio_cd
GPIO number of card detect signal.

gpio_num_t gpio_wp
GPIO number of write protect signal.

gpio_num_t gpio_int
GPIO number of interrupt line (input) for SDIO card.

Macros

SDSPI_DEFAULT_HOST
Default sdmmc_host_t structure initializer for SD over SPI driver.
Uses SPI mode and max frequency set to 20MHz
‘slo’t should be set to an sdspi device initialized by sdspi_host_init_device().

SDSPI_SLOT_NO_CS
indicates that card select line is not used

SDSPI_SLOT_NO_CD
indicates that card detect line is not used

SDSPI_SLOT_NO_WP
indicates that write protect line is not used

SDSPI_SLOT_NO_INT
indicates that interrupt line is not used

SDSPI_DEVICE_CONFIG_DEFAULT()
Macro defining default configuration of SD SPI device.
Type Definitions

typedef int sdspi_dev_handle_t
  Handle representing an SD SPI device.

2.6.18 SDIO Card Slave Driver

Overview

The ESP32 SDIO Card peripherals (Host, Slave) shares two sets of pins as below table. The first set is usually occupied by SPI0 bus which is responsible for the SPI flash holding the code to run. This means SDIO slave driver can only runs on the second set of pins while SDIO host is not using it.

The SDIO slave can run under 3 modes: SPI, 1-bit SD and 4-bit SD modes, which is detected automatically by the hardware. According to the SDIO specification, CMD and DAT0-3 lines should be pulled up no matter in 1-bit, 4-bit or SPI mode.

<table>
<thead>
<tr>
<th>Connections</th>
<th>Pin Name</th>
<th>Corresponding pins in SPI mode</th>
<th>Slot1</th>
<th>Slot2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CLK</td>
<td>SCLK</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>CMD</td>
<td>MOSI</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>DAT0</td>
<td>MISO</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>DAT1</td>
<td>Interrupt</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>DAT2</td>
<td>N.C. (pullup)</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>DAT3</td>
<td>#CS</td>
<td>10</td>
<td>13</td>
</tr>
</tbody>
</table>

- 1-bit SD mode: Connect CLK, CMD, DAT0, DAT1 pins and the ground.
- 4-bit SD mode: Connect all pins and the ground.
- SPI mode: Connect SCLK, MOSI, MISO, Interrupt, #CS pins and the ground.

备注： Please check if CMD and DATAlines D0-D3 of the card are properly pulled up by 10 Kohm resistors. This should be ensured even in 1-bit mode or SPI mode. Most official modules don’t offer these pullups internally. If you are using official development boards, check Overview of Compatibility to see whether your development boards have such pullups.

备注： Most official modules have conflicts on strapping pins with the SDIO slave function. If you are using a ESP32 module with 3.3 V flash inside, you have to burn the EFUSE when you are developing on the module for the first time. See Overview of Compatibility to see how to make your modules compatible with the SDIO.

Here is a list for modules/kits with 3.3 V flash:

- Kits: ESP32-PICO-KIT, ESP32-DevKitC (till v4), ESP32-WROVER-KIT (v4.1 (also known as ESP32-WROVER-KIT-VB), v2, v1 (also known as DevKitJ v1))

You can tell the version of your ESP32-WROVER-KIT version from the module on it: v4.1 are with ESP32-WROVER-B modules, v3 are with ESP32-WROVER modules, while v2 and v1 are with ESP32-WROOM-32 modules.

Refer to SD Pull-up Requirements for more technical details of the pullups.

The host initialize the slave into SD mode by first sending CMD0 with DAT3 pin high, or in SPI mode by sending CMD0 with CS pin (the same pin as DAT3) low.
After the initialization, the host can enable the 4-bit SD mode by writing CCCR register 0x07 by CMD52. All the bus detection process are handled by the slave peripheral.

The host has to communicate with the slave by an ESP-slave-specific protocol. The slave driver offers 3 services over Function 1 access by CMD52 and CMD53: (1) a sending FIFO and a receiving FIFO, (2) 52 8-bit R/W registers shared by host and slave, (3) 16 interrupt sources (8 from host to slave, and 8 from slave to host).

**Terminology** The SDIO slave driver uses the following terms:

- **Transfer**: a transfer is always started by a command token from the host, and may contain a reply and several data blocks. ESP32 slave software is based on transfers.
- **Sending**: slave to host transfers.
- **Receiving**: host to slave transfers.

**Remark**: Register names in ESP32 Technical Reference Manual > SDIO Slave Controller [PDF] are oriented from the point of view of the host, i.e. ‘rx’ registers refer to sending, while ‘tx’ registers refer to receiving. We’re not using tx or rx in the driver to avoid ambiguities.

- **FIFO**: specific address in Function 1 that can be access by CMD53 to read/write large amount of data. The address is related to the length requested to read from/write to the slave in a single transfer: requested length = 0x1F800-address.
- **Ownership**: When the driver takes ownership of a buffer, it means the driver can randomly read/write the buffer (usually via DMA). The application should not read/write the buffer until the ownership is returned to the application. If the application reads from a buffer owned by a receiving driver, the data read can be random; if the application writes to a buffer owned by a sending driver, the data sent may be corrupted.
- **Requested length**: The length requested in one transfer determined by the FIFO address.
- **Transfer length**: The length requested in one transfer determined by the CMD53 byte/block count field.

**Remark**: Requested length is different from the transfer length. ESP32 slave DMA base on the requested length rather than the transfer length. The transfer length should be no shorter than the requested length, and the rest part will be filled with 0 (sending) or discard (receiving).

- **Receiving buffer size**: The buffer size is pre-defined between the host and the slave before communication starts. Slave application has to set the buffer size during initialization by the recv_buffer_size member of sdio_slave_config_t.
- **Interrupts**: the esp32 slave support interrupts in two directions: from host to slave (called slave interrupts below) and from slave to host (called host interrupts below). See more in Interrupts.
- **Registers**: specific address in Function 1 access by CMD52 or CMD53.

**Communication with ESP SDIO Slave** The host should initialize the ESP32 SDIO slave according to the standard SDIO initialization process (Sector 3.1.2 of SDIO Simplified Specification), which is described briefly in ESP SDIO Slave Initialization.

Furthermore, there’s an ESP32-specific upper-level communication protocol upon the CMD52/CMD53 to Func 1. Please refer to ESP SDIO Slave Protocol. There is also a component ESP Serial Slave Link for ESP32 master to communicate with ESP32 SDIO slave, see example peripherals/sdio when programming your host.

**Interrupts** There are interrupts from host to slave, and from slave to host to help communicating conveniently.

**Slave Interrupts** The host can interrupt the slave by writing any one bit in the register 0x08D. Once any bit of the register is set, an interrupt is raised and the SDIO slave driver calls the callback function defined in the slave_intr_cb member in the sdio_slave_config_t structure.
The callback function is called in the ISR, do not use any delay, loop or spinlock in the callback.

There’s another set of functions can be used. You can call `sdio_slave_wait_int` to wait for an interrupt within a certain time, or call `sdio_slave_clear_int` to clear interrupts from host. The callback function can work with the wait functions perfectly.

**Host Interrupts** The slave can interrupt the host by an interrupt line (at certain time) which is level sensitive. When the host see the interrupt line pulled down, it may read the slave interrupt status register, to see the interrupt source. Host can clear interrupt bits, or choose to disable a interrupt source. The interrupt line will hold active until all the sources are cleared or disabled.

There are several dedicated interrupt sources as well as general purpose sources. see `sdio_slave_hostint_t` for more information.

**Shared Registers** There are 52 8-bit R/W shared registers to share information between host and slave. The slave can write or read the registers at any time by `sdio_slave_read_reg` and `sdio_slave_write_reg`. The host can access (R/W) the register by CMD52 or CMD53.

**Receiving FIFO** When the host is going to send the slave some packets, it has to check whether the slave is ready to receive by reading the buffer number of slave.

To allow the host sending data to the slave, the application has to load buffers to the slave driver by the following steps:

1. Register the buffer by calling `sdio_slave_recv_register_buf`, and get the handle of the registered buffer. The driver will allocate memory for the linked-list descriptor needed to link the buffer onto the hardware. The size of these buffers should equal to the Receiving buffer size.
2. Load buffers onto the driver by passing the buffer handle to `sdio_slave_recv_load_buf`.
3. Get the received data by calling `sdio_slave_recv` or `sdio_slave_recv_packet`. If non-blocking call is needed, set `wait=0`.

The difference between two APIs is that, `sdio_slave_recv_packet` gives more information about packet, which can consist of several buffers. When `ESP_ERR_NOT_FINISHED` is returned by this API, you should call this API iteratively until the return value is `ESP_OK`. All the continuous buffers returned with `ESP_ERR_NOT_FINISHED`, together with the last buffer returned with `ESP_OK`, belong to one packet from the host. Call `sdio_slave_recv_get_buf` to get the address of the received data, and the actual length received in each buffer. The packet length is the sum of received length of all the buffers in the packet.

If the host never send data longer than the Receiving buffer size, or you don’t care about the packet boundary (e.g. the data is only a byte stream), you can call the simpler version `sdio_slave_recv` instead.

4. Pass the handle of processed buffer back to the driver by `sdio_recv_load_buf` again.

**Sending FIFO** Each time the slave has data to send, it raises an interrupt and the host will request for the packet length. There are two sending modes:

- Stream Mode: when a buffer is loaded to the driver, the buffer length will be counted into the packet length requested by host in the incoming communications. Regardless previous packets are sent or not. This means the host can get data of several buffers in one transfer.
- Packet Mode: the packet length is updated packet by packet, and only when previous packet is sent. This means that the host can only get data of one buffer in one transfer.
To avoid overhead from copying data, the driver itself doesn’t have any buffer inside. Namely, the DMA takes data directly from the buffer provided by the application. The application should not touch the buffer until the sending is finished.

The sending mode can be set in the `sending_mode` member of `sdio_slave_config_t`, and the buffer numbers can be set in the `send_queue_size`. All the buffers are restricted to be no larger than 4092 bytes. Though in the stream mode several buffers can be sent in one transfer, each buffer is still counted as one in the queue.

The application can call `sdio_slave_transmit` to send packets. In this case the function returns when the transfer is successfully done, so the queue is not fully used. When higher efficiency is required, the application can use the following functions instead:

1. Pass buffer information (address, length, as well as an `arg` indicating the buffer) to `sdio_slave_send_queue`. If non-blocking call is needed, set `wait=0`. If the `wait` is not `portMAX_DELAY` (wait until success), the application has to check the result to know whether the data is put in to the queue or discard.
2. Call `sdio_slave_send_get_finished` to get and deal with a finished transfer. A buffer should be keep unmodified until returned from `sdio_slave_send_get_finished`. This means the buffer is actually sent to the host, rather than just staying in the queue.

There are several ways to use the `arg` in the queue parameter:

1. Directly point `arg` to a dynamic-allocated buffer, and use the `arg` to free it when transfer finished.
2. Wrap transfer informations in a transfer structure, and point `arg` to the structure. You can use the structure to do more things like:

   ```c
   typedef struct {
       uint8_t* buffer;
       size_t size;
       int id;
    } sdio_transfer_t;

    // and send as:
    sdio_transfer_t trans = {
        .buffer = ADDRESS_TO_SEND,
        .size = 8,
        .id = 3, // the 3rd transfer so far
    };
    sdio_slave_send_queue(trans.buffer, trans.size, &trans, portMAX_DELAY);

    //... maybe more transfers are sent here

    // and deal with finished transfer as:
    sdio_transfer_t* arg = NULL;
    sdio_slave_send_get_finished((void*)arg, portMAX_DELAY);
    ESP_LOGI("tag", "(id) successfully send %d bytes of %p", arg->id, arg->size,...
    some_post_callback(arg); // do more things

   3. Working with the receiving part of this driver, point `arg` to the receive buffer handle of this buffer. So that we can directly use the buffer to receive data when it’s sent:

   ```c
   uint8_t buffer[256]={1,2,3,4,5,6,7,8};
   sdio_slave_buf_handle_t handle = sdio_slave_recv_register_buf(buffer);
   sdio_slave_send_queue(buffer, 8, handle, portMAX_DELAY);

   //... maybe more transfers are sent here

   // and load finished buffer to receive as
   sdio_slave_buf_handle_t handle = NULL;
   sdio_slave_send_get_finished((void*)handle, portMAX_DELAY);
   sdio_slave_recv_load_buf(handle);
   ```
More about this, see peripherals/sdio.

Application Example

Slave/master communication: peripherals/sdio.

API Reference

Header File

- components/hal/include/hal/sdio_slave_types.h

Enumerations

enum sdio_slave_hostint_t

Mask of interrupts sending to the host.

Values:

enumerator SDIO_SLAVE_HOSTINT_BIT0

General purpose interrupt bit 0.

ee

enumerator SDIO_SLAVE_HOSTINT_BIT1

e

enumerator SDIO_SLAVE_HOSTINT_BIT2

e

enumerator SDIO_SLAVE_HOSTINT_BIT3

e

enumerator SDIO_SLAVE_HOSTINT_BIT4

e

enumerator SDIO_SLAVE_HOSTINT_BIT5

e

enumerator SDIO_SLAVE_HOSTINT_BIT6

e

enumerator SDIO_SLAVE_HOSTINT_BIT7

e

enumerator SDIO_SLAVE_HOSTINT_SEND_NEW_PACKET

New packet available.

enum sdio_slave_timing_t

Timing of SDIO slave.

Values:

enumerator SDIO_SLAVE_TIMING_PSEND_PSAMPLE

Send at posedge, and sample at posedge. Default value for HS mode. If `:macro:SDIO_SLAVE_FLAG_HIGH_SPEED` is specified in :cpp:class:sdio_slave_config_t, this should be selected. Normally there’s no problem using this to work in DS mode.
enumerator SDIO_SLAVE_TIMING_NSEND_PSAMPLE
Send at negedge, and sample at posedge. Default value for DS mode and below. If \texttt{SDIO_SLAVE_FLAG_DEFAULT_SPEED} is specified in \texttt{sdio_slave_config_t}, this should be selected.

denumerator SDIO_SLAVE_TIMING_PSEND_NSAMPLE
Send at posedge, and sample at negedge.

denumerator SDIO_SLAVE_TIMING_NSEND_NSAMPLE
Send at negedge, and sample at negedge.

enum sdio_slave_sending_mode_t
Configuration of SDIO slave mode.

Values:

denumerator SDIO_SLAVE_SEND_STREAM
Stream mode, all packets to send will be combined as one if possible.

denumerator SDIO_SLAVE_SEND_PACKET
Packet mode, one packets will be sent one after another (only increase packet_len if last packet sent).

Header File

- components/driver/include/driver/sdio_slave.h

Functions

\texttt{esp_err_t sdio_slave_init\_initialize(sdio_slave_config_t *config)}

Initialize the sdio slave driver

参数 config - Configuration of the sdio slave driver.

返回
- ESP_OK if success
- ESP_ERR_NOT_FOUND if no free interrupt found.
- ESP_ERR_INVALID_STATE if already initialized.
- ESP_ERR_NO_MEM if fail due to memory allocation failed.
- ESP_OK if success

void sdio_slave_deinit (void)

De-initialize the sdio slave driver to release the resources.

\texttt{esp_err_t sdio_slave_start (void)}

Start hardware for sending and receiving, as well as set the IOREADY1 to 1.

备注: The driver will continue sending from previous data and PKT_LEN counting. keep data received as well as start receiving from current TOKEN1 counting. See \texttt{sdio_slave_reset}.

返回
- ESP_ERR_INVALID_STATE if already started.
- ESP_OK otherwise.
void `sdio_slave_stop` (void)
Stop hardware from sending and receiving, also set IREADY1 to 0.

**备注:** this will not clear the data already in the driver, and also not reset the PKT_LEN and TOKEN1 counting. Call `sdio_slave_reset` to do that.

```c
esp_err_t `sdio_slave_reset` (void)
```
Clear the data still in the driver, as well as reset the PKT_LEN and TOKEN1 counting.

**备注:** always return ESP_OK.

```c
sdio_slave_buf_handle_t `sdio_slave_recv_register_buf` (uint8_t* start)
```
Register buffer used for receiving. All buffers should be registered before used, and then can be used (again) in the driver by the handle returned.

**备注:** The driver will use and only use the amount of space specified in the `recv_buffer_size` member set in the `sdio_slave_config_t`. All buffers should be larger than that. The buffer is used by the DMA, so it should be DMA capable and 32-bit aligned.

- **参数** `start` – The start address of the buffer.
- **返回** The buffer handle if success, otherwise NULL.

```c
esp_err_t `sdio_slave_recv_unregister_buf` (sdio_slave_buf_handle_t handle)
```
Unregister buffer from driver, and free the space used by the descriptor pointing to the buffer.

- **参数** `handle` – Handle to the buffer to release.
- **返回** ESP_OK if success, ESP_ERR_INVALID_ARG if the handle is NULL or the buffer is being used.

```c
esp_err_t `sdio_slave_recv_load_buf` (sdio_slave_buf_handle_t handle)
```
Load buffer to the queue waiting to receive data. The driver takes ownership of the buffer until the buffer is returned by `sdio_slave_send_get_finished` after the transaction is finished.

- **参数** `handle` – Handle to the buffer ready to receive data.

- **返回**
  - ESP_ERR_INVALID_ARG if invalid handle or the buffer is already in the queue. Only after the buffer is returned by `sdio_slave_recv` can you load it again.
  - ESP_OK if success

```c
esp_err_t `sdio_slave_recv_packet` (sdio_slave_buf_handle_t *handle_ret, TickType_t wait)
```
Get buffer of received data if exist with packet information. The driver returns the ownership of the buffer to the app.

When you see return value is ESP_ERR_NOT_FINISHED, you should call this API iteratively until the return value is ESP_OK. All the continuous buffers returned with ESP_ERR_NOT_FINISHED, together with the last buffer returned with ESP_OK, belong to one packet from the host.

You can call simpler `sdio_slave_recv` instead, if the host never send data longer than the Receiving buffersize, or you don’t care about the packet boundary (e.g. the data is only a byte stream).

**备注:** Call `sdio_slave_load_buf` with the handle to re-load the buffer onto the link list, and receive with the same buffer again. The address and length of the buffer got here is the same as got from `sdio_slave_get_buffer`.

- **参数**
  - `handle_ret` – Handle of the buffer holding received data. Use this handle in `sdio_slave_recv_load_buf()` to receive in the same buffer again.
**Chapter 2. API**

- **wait** - Time to wait before data received.

**返回**
- ESP_ERR_INVALID_ARG if handle_ret is NULL
- ESP_ERR_TIMEOUT if timeout before receiving new data
- ESP_ERR_NOT_FINISHED if returned buffer is not the end of a packet from the host.
  
  should call this API again until the end of a packet
- ESP_OK if success

```c
esp_err_t sdio_slave_recv (sdio_slave_buf_handle_t *handle_ret, uint8_t **out_addr, size_t *out_len,
TickType_t wait)
```

Get received data if exist. The driver returns the ownership of the buffer to the app.

**备注:** Call sdio_slave_load_buf with the handle to re-load the buffer onto the link list, and receive with the same buffer again. The address and length of the buffer got here is the same as got from sdio_slave_get_buffer.

**参数**
- **handle_ret** - Handle to the buffer holding received data. Use this handle in sdio_slave_recv_load_buf to receive in the same buffer again.
- **out_addr** – [out] Output of the start address, set to NULL if not needed.
- **out_len** – [out] Actual length of the data in the buffer, set to NULL if not needed.
- **wait** – Time to wait before data received.

**返回**
- ESP_ERR_INVALID_ARG if handle_ret is NULL
- ESP_ERR_TIMEOUT if timeout before receiving new data
- ESP_OK if success

```c
uint8_t *sdio_slave_recv_get_buf (sdio_slave_buf_handle_t handle, size_t *len_o)
```

Retrieve the buffer corresponding to a handle.

**参数**
- **handle** - Handle to get the buffer.
- **len_o** – Output of buffer length

**返回**
- buffer address if success, otherwise NULL.

```c
esp_err_t sdio_slave_send_queue (uint8_t *addr, size_t len, void *arg, TickType_t wait)
```

Put a new sending transfer into the send queue. The driver takes ownership of the buffer until the buffer is returned by sdio_slave_send_get_finished after the transaction is finished.

**参数**
- **addr** – Address for data to be sent. The buffer should be DMA capable and 32-bit aligned.
- **len** – Length of the data, should not be longer than 4092 bytes (may support longer in the future).
- **arg** – Argument to returned in sdio_slave_send_get_finished. The argument can be used to indicate which transaction is done, or as a parameter for a callback. Set to NULL if not needed.
- **wait** – Time to wait if the buffer is full.

**返回**
- ESP_ERR_INVALID_ARG if the length is not greater than 0.
- ESP_ERR_TIMEOUT if the queue is still full until timeout.
- ESP_OK if success.

```c
esp_err_t sdio_slave_send_get_finished (void **out_arg, TickType_t wait)
```

Return the ownership of a finished transaction.

**参数**
- **out_arg** – Argument of the finished transaction. Set to NULL if unused.
- **wait** – Time to wait if there’s no finished sending transaction.
**Chapter 2. API 参考**

返回 ESP_ERR_TIMEOUT if no transaction finished, or ESP_OK if succeed.

`esp_err_t sdio_slave_transmit (uint8_t *addr, size_t len)`

Start a new sending transfer, and wait for it (blocked) to be finished.

参数
- `addr` - Start address of the buffer to send
- `len` - Length of buffer to send.

返回
- ESP_ERR_INVALID_ARG if the length of descriptor is not greater than 0.
- ESP_ERR_TIMEOUT if the queue is full or host do not start a transfer before timeout.
- ESP_OK if success.

`uint8_t sdio_slave_read_reg (int pos)`

Read the spi slave register shared with host.

备注: register 28 to 31 are reserved for interrupt vector.

参数 `pos` - register address, 0-27 or 32-63.
返回 value of the register.

`esp_err_t sdio_slave_write_reg (int pos, uint8_t reg)`

Write the spi slave register shared with host.

备注: register 29 and 31 are used for interrupt vector.

参数
- `pos` - register address, 0-11, 14-15, 18-19, 24-27 and 32-63, other address are reserved.
- `reg` - the value to write.

返回 ESP_ERR_INVALID_ARG if address wrong, otherwise ESP_OK.

`sdio_slave_hostint_t sdio_slave_get_host_intena (void)`

Get the interrupt enable for host.

返回 the interrupt mask.

`void sdio_slave_set_host_intena (sdio_slave_hostint_t mask)`

Set the interrupt enable for host.

参数 `mask` - Enable mask for host interrupt.

`esp_err_t sdio_slave_send_host_int (uint8_t pos)`

Interrupt the host by general purpose interrupt.

参数 `pos` - Interrupt num, 0-7.

返回
- ESP_ERR_INVALID_ARG if interrupt num error
- ESP_OK otherwise

`void sdio_slave_clear_host_int (sdio_slave_hostint_t mask)`

Clear general purpose interrupt to host.

参数 `mask` - Interrupt bits to clear, by bit mask.

`esp_err_t sdio_slave_wait_int (int pos, TickType_t wait)`

Wait for general purpose interrupt from host.

备注: this clears the interrupt at the same time.
### Structures

```c
struct sdio_slave_config_t
```

Configuration of SDIO slave.

### Public Members

- `sdio_slave_timing_t timing`
  - timing of `sdio_slave`. see `sdio_slave_timing_t`.

- `sdio_slave_sending_mode_t sending_mode`
  - mode of `sdio_slave`. `SDIO_SLAVE_MODE_STREAM` if the data needs to be sent as much as possible; `SDIO_SLAVE_MODE_PACKET` if the data should be sent in packets.

- `int send_queue_size`
  - max buffers that can be queued before sending.

- `size_t recv_buffer_size`
  - If buffer size is too small, it costs more CPU time to handle larger number of buffers. If buffer size is too large, the space larger than the transaction length is left blank but still counts a buffer, and the buffers are easily run out. Should be set according to length of data really transferred. All data that do not fully fill a buffer is still counted as one buffer. E.g. 10 bytes data costs 2 buffers if the size is 8 bytes per buffer. Buffer size of the slave pre-defined between host and slave before communication. All receive buffer given to the driver should be larger than this.

- `sdio_event_cb_t event_cb`
  - when the host interrupts slave, this callback will be called with interrupt number (0-7).

- `uint32_t flags`
  - Features to be enabled for the slave, combinations of `SDIO_SLAVE_FLAG_*`.

### Macros

- **SDIO_SLAVE_RECV_MAX_BUFFER**
- **SDIO_SLAVE_FLAG_DAT2_DISABLED**
  - It is required by the SD specification that all 4 data lines should be used and pulled up even in 1-bit mode or SPI mode. However, as a feature, the user can specify this flag to make use of DAT2 pin in 1-bit mode. Note that the host cannot read CCCR registers to know we don’t support 4-bit mode anymore, please do this at your own risk.

- **SDIO_SLAVE_FLAG_HOST_INTR_DISABLED**
  - The DAT1 line is used as the interrupt line in SDIO protocol. However, as a feature, the user can specify this flag to make use of DAT1 pin of the slave in 1-bit mode. Note that the host has to do polling to the interrupt registers to know whether there are interrupts from the slave. And it cannot read CCCR registers to know we don’t support 4-bit mode anymore, please do this at your own risk.
Chapter 2. API

SDIO_SLAVE_FLAG_INTERNAL_PULLUP

Enable internal pullups for enabled pins. It is required by the SD specification that all the 4 data lines should be pulled up even in 1-bit mode or SPI mode. Note that the internal pull-ups are not sufficient for stable communication, please do connect external pull-ups on the bus. This is only for example and debug use.

SDIO_SLAVE_FLAG_DEFAULT_SPEED

Disable the highspeed support of the hardware.

SDIO_SLAVE_FLAG_HIGH_SPEED

Enable the highspeed support of the hardware. This is the default option. The host will see highspeed capability, but the mode actually used is determined by the host.

Type Definitions

typedef void (*sdio_event_cb_t)(uint8_t event)

typedef void *sdio_slave_buf_handle_t

Handle of a receive buffer, register a handle by calling `sdio_slave_recv_register_buf`. Use the handle to load the buffer to the driver, or call `sdio_slave_recv_unregister_buf` if it is no longer used.

2.6.19 Sigma-Delta Modulation (SDM)

Introduction

ESP32 has a second-order sigma-delta modulator, which can generate independent PDM pulses to multiple channels. Please refer to the TRM to check how many hardware channels are available.1

Typically, a Sigma-Delta modulated channel can be used in scenarios like:

- LED dimming
- Simple DAC (8-bit), with the help of an active RC low-pass filter
- Class D amplifier, with the help of a half-bridge or full-bridge circuit plus an LC low-pass filter

Functional Overview

The following sections of this document cover the typical steps to install and operate a SDM channel:

- **Resource Allocation** - covers which parameters should be set up to get a channel handle and how to recycle the resources when it finishes working.
- **Enable and Disable Channel** - covers how to enable and disable the channel.
- **Set Equivalent Duty Cycle** - describes how to set the equivalent duty cycle of the PDM pulses.
- **Power Management** - describes how different source clock selections can affect power consumption.
- **IRAM Safe** - lists which functions are supposed to work even when the cache is disabled.
- **Thread Safety** - lists which APIs are guaranteed to be thread safe by the driver.
- **Kconfig Options** - lists the supported Kconfig options that can be used to make a different effect on driver behavior.

---

1 Different ESP chip series might have different numbers of SDM channels. Please refer to Chapter GPIO and IOMUX in ESP32 Technical Reference Manual for more details. The driver won’t forbid you from applying for more channels, but it will return error when all available hardware resources are used up. Please always check the return value when doing resource allocation (e.g. `sdm_new_channel()`).
Resource Allocation  A SDM channel is represented by `sdm_channel_handle_t`. Each channel is capable to output the binary, hardware generated signal with the sigma-delta modulation. The driver manages all available channels in a pool, so that users don’t need to manually assign a fixed channel to a GPIO.

To install a SDM channel, you should call `sdm_new_channel()` to get a channel handle. Channel specific configurations are passed in the `sdm_config_t` structure:

- `sdm_config_t::gpio_num` sets the GPIO that the PDM pulses will output from
- `sdm_config_t::clk_src` selects the source clock for the SDM module. Note that, all channels should select the same clock source.
- `sdm_config_t::sample_rate_hz` sets the sample rate of the SDM module.
- `sdm_config_t::invert_out` sets whether to invert the output signal.
- `sdm_config_t::io_loop_back` is for debugging purposes only. It enables both the GPIO’s input and output ability through the GPIO matrix peripheral.

The function `sdm_new_channel()` can fail due to various errors such as insufficient memory, invalid arguments, etc. Specifically, when there are no more free channels (i.e. all hardware SDM channels have been used up), then `ESP_ERR_NOT_FOUND` will be returned.

If a previously created SDM channel is no longer required, you should recycle it by calling `sdm_del_channel()`. It allows the underlying HW channel to be used for other purposes. Before deleting a SDM channel handle, you should disable it by `sdm_channel_disable()` in advance or make sure it has not enabled yet by `sdm_channel_enable()`.

Creating a SDM Channel with Sample Rate of 1MHz

```
sdm_channel_handle_t chan = NULL;
sdm_config_t config = {
    .clk_src = SDM_CLK_SRC_DEFAULT,
    .sample_rate_hz = 1 * 1000 * 1000,
    .gpio_num = 0,
};
ESP_ERROR_CHECK(sdm_new_channel(&config, &chan));
```

Enable and Disable Channel  Before doing further IO control to the SDM channel, you should enable it first, by calling `sdm_channel_enable()`. Internally, this function will:

- switch the channel state from init to enable
- acquire a proper power management lock is a specific clock source (e.g. APB clock) is selected. See also Power management for more information.

On the contrary, calling `sdm_channel_disable()` will do the opposite, that is, put the channel back to the init state and release the power management lock.

Set Equivalent Duty Cycle  For the output PDM signals, the duty cycle refers to the percentage of high level cycles to the whole statistical period. The average output voltage from the channel is calculated by

\[
V_{out} = \frac{VDD_{IO}}{256} \times \text{duty} + \frac{VDD_{IO}}{2}
\]

Thus the range of the duty input parameter of `sdm_channel_set_duty()` is from -128 to 127 (eight bit signed integer). For example, if zero value is set, then the output signal’s duty will be about 50%.

Power Management  When power management is enabled (i.e. `CONFIG_PM_ENABLE` is on), the system will adjust the APB frequency before going into light sleep, thus potentially changing the sample rate of the sigma-delta modulator.

However, the driver can prevent the system from changing APB frequency by acquiring a power management lock of type `ESP_PM_APB_FREQ_MAX`. Whenever the driver creates a SDM channel instance that has selected `SDM_CLK_SRC_APB` as its clock source, the driver will guarantee that the power management lock is acquired when enable the channel by `sdm_channel_enable()`. Likewise, the driver releases the lock when `sdm_channel_disable()` is called for that channel.
**IRAM Safe** There’s a Kconfig option `CONFIG_SDM_CTRL_FUNC_IN_IRAM` that can put commonly used IO control functions into IRAM as well. So that these functions can also be executable when the cache is disabled. These IO control functions are listed as follows:

- `sdm_channel_set_duty()`

**Thread Safety** The factory function `sdm_new_channel()` is guaranteed to be thread safe by the driver, which means, user can call it from different RTOS tasks without protection by extra locks. The following functions are allowed to run under ISR context, the driver uses a critical section to prevent them being called concurrently in both task and ISR.

- `sdm_channel_set_duty()`

Other functions that take the `sdm_channel_handle_t` as the first positional parameter, are not treated as thread safe. Which means the user should avoid calling them from multiple tasks.

**Kconfig Options**

- `CONFIG_SDM_CTRL_FUNC_IN_IRAM` controls where to place the SDM channel control functions (IRAM or Flash), see IRAM Safe for more information.
- `CONFIG_SDM_ENABLE_DEBUG_LOG` is used to enabled the debug log output. Enable this option will increase the firmware binary size.

**Convert to analog signal (Optional)**

Typically, if the sigma-delta signal is connected to an LED, you don’t have to add any filter between them (because our eyes are a low pass filter naturally). However, if you want to check the real voltage or watch the analog waveform, you need to design an analog low pass filter. Also, it is recommended to use an active filter instead of a passive filter to gain better isolation and not lose too much voltage.

For example, you can take the following Sallen-Key topology Low Pass Filter as a reference.

![Sallen-Key Low Pass Filter](image)

图 19: Sallen-Key Low Pass Filter
Application Example

- LED driven by a GPIO that is modulated with Sigma-Delta: peripherals/sigma_delta.

API Reference

Header File

- components/driver/include/driver/sdm.h

Functions

`esp_err_t sdm_new_channel (const sdm_config_t *config, sdm_channel_handle_t *ret_chan)`
Create a new Sigma Delta channel.

参数
- `config` - [in] SDM configuration
- `ret_chan` - [out] Returned SDM channel handle

返回
- ESP_OK: Create SDM channel successfully
- ESP_ERR_INVALID_ARG: Create SDM channel failed because of invalid argument
- ESP_ERR_NO_MEM: Create SDM channel failed because out of memory
- ESP_ERR_NOT_FOUND: Create SDM channel failed because all channels are used up and no more free one
- ESP_FAIL: Create SDM channel failed because of other error

`esp_err_t sdm_del_channel (sdm_channel_handle_t chan)`
Delete the Sigma Delta channel.

参数 `chan` - [in] SDM channel created by sdm_new_channel

返回
- ESP_OK: Delete the SDM channel successfully
- ESP_ERR_INVALID_ARG: Delete the SDM channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Delete the SDM channel failed because the channel is not in init state
- ESP_FAIL: Delete the SDM channel failed because of other error

`esp_err_t sdm_channel_enable (sdm_channel_handle_t chan)`
Enable the Sigma Delta channel.

备注: This function will transit the channel state from init to enable.

备注: This function will acquire a PM lock, if a specific source clock (e.g. APB) is selected in the sdm_config_t, while CONFIG_PM_ENABLE is enabled.

参数 `chan` - [in] SDM channel created by sdm_new_channel

返回
- ESP_OK: Enable SDM channel successfully
- ESP_ERR_INVALID_ARG: Enable SDM channel failed because of invalid argument
- ESP_ERR_INVALID_STATE: Enable SDM channel failed because the channel is already enabled
- ESP_FAIL: Enable SDM channel failed because of other error

`esp_err_t sdm_channel_disable (sdm_channel_handle_t chan)`
Disable the Sigma Delta channel.
This function will do the opposite work to the `sdm_channel_enable()` function.

### Parameters
- **chan** - [in] SDM channel created by `sdm_new_channel`

### Returs
- `ESP_OK`: Disable SDM channel successfully
- `ESP_ERR_INVALID_ARG`: Disable SDM channel failed because of invalid argument
- `ESP_ERR_INVALID_STATE`: Disable SDM channel failed because the channel is not enabled yet
- `ESP_FAIL`: Disable SDM channel failed because of other error

```c
esp_err_t sdm_channel_set_duty(sdm_channel_handle_t chan, int8_t duty)
```

Set the duty cycle of the PDM output signal.

### Notes
- For PDM signals, duty cycle refers to the percentage of high level cycles to the whole statistical period. The average output voltage could be \[ V_{out} = \frac{V_{DD, IO}}{256} \times \text{duty} + \frac{V_{DD, IO}}{2} \].
- If the duty is set to zero, the output signal is like a 50% duty cycle square wave, with a frequency around \( \frac{\text{sample_rate}_{hz}}{4} \).
- The duty is proportional to the equivalent output voltage after a low-pass-filter.
- This function is allowed to run within ISR context
- This function will be placed into IRAM if `CONFIG_SDM_CTRL_FUNC_IN_IRAM` is on, so that it’s allowed to be executed when Cache is disabled

### Notes
- **chan** - [in] SDM channel created by `sdm_new_channel`
- **duty** - [in] Equivalent duty cycle of the PDM output signal, ranges from -128 to 127. But the range of [-90, 90] can provide a better randomness.

### Returns
- `ESP_OK`: Set duty cycle successfully
- `ESP_ERR_INVALID_ARG`: Set duty cycle failed because of invalid argument
- `ESP_FAIL`: Set duty cycle failed because of other error

### Structures

```c
struct sdm_config_t
```

Sigma Delta channel configuration.

### Public Members

```c
int gpios_num
```

GPIO number
Chapter 2. API

```c
sdm_clock_source_t clk_src

Clock source
```

```c
uint32_t sample_rate_hz

Sample rate in Hz, it determines how frequent the modulator outputs a pulse
```

```c
uint32_t invert_out

Whether to invert the output signal
```

```c
uint32_t io_loop_back

For debug/test, the signal output from the GPIO will be fed to the input path as well
```

```c
struct sdm_config_t::[anonymous] flags

Extra flags
```

**Type Definitions**

typedef struct sdm_channel_t *sdm_channel_handle_t

Type of Sigma Delta channel handle.

**Header File**

- components/hal/include/hal/sdm_types.h

**Type Definitions**

typedef soc_periph_sdm_clk_src_t sdm_clock_source_t

### 2.6.20 SPI Master Driver

SPI Master driver is a program that controls ESP32’s SPI peripherals while they function as masters.

**Overview of ESP32’s SPI peripherals**

ESP32 integrates 4 SPI peripherals.

- SPI0 and SPI1 are used internally to access the ESP32’s attached flash memory. Both controllers share the same SPI bus signals, and there is an arbiter to determine which can access the bus. There are quite a few limitations when using SPI Master driver on the SPI1 bus, see Notes on Using the SPI Master driver on SPI1 Bus.

- SPI2 and SPI3 are general purpose SPI controllers, sometimes referred to as HSPI and VSPI, respectively. They are open to users. SPI2 and SPI3 have independent bus signals with the same respective names. Each bus has three CS lines to drive up to same number of SPI slaves.

**Terminology**

The terms used in relation to the SPI master driver are given in the table below.
# Chapter 2. API

## Host

The SPI controller peripheral inside ESP32 that initiates SPI transmissions over the bus, and acts as an SPI Master.

## Device

SPI slave device. An SPI bus may be connected to one or more Devices. Each Device shares the MOSI, MISO and SCLK signals but is only active on the bus when the Host asserts the Device’s individual CS line.

## Bus

A signal bus, common to all Devices connected to one Host. In general, a bus includes the following lines: MOSI, MOSI, SCLK, one or more CS lines, and, optionally, QUADWP and QUADHD. So Devices are connected to the same lines, with the exception that each Device has its own CS line. Several Devices can also share one CS line if connected in the daisy-chain manner.

## MOSI

Master Out, Slave In, a.k.a. D. Data transmission from a Host to Device. Also data0 signal in Octal/OPI mode.

## MISO

Master In, Slave Out, a.k.a. Q. Data transmission from a Device to Host. Also data1 signal in Octal/OPI mode.

## SCLK

Serial Clock. Oscillating signal generated by a Host that keeps the transmission of data bits in sync.

## CS

Chip Select. Allows a Host to select individual Device(s) connected to the bus in order to send or receive data.

## QUADWP

Protect signal. Used for 4-bit (qio/qout) transactions. Also for data2 signal in Octal/OPI mode.

## QUADHD

Signal. Used for 4-bit (qio/qout) transactions. Also for data3 signal in Octal/OPI mode.

## DATA4

Data4 signal in Octal/OPI mode.

## DATA5

Data5 signal in Octal/OPI mode.

## DATA6

Data6 signal in Octal/OPI mode.

## DATA7

Data7 signal in Octal/OPI mode.

### Assertion

The action of activating a line.

### De-assertion

The action of returning the line back to inactive (back to idle) status.

### Transaction

One instance of a Host asserting a CS line, transferring data to and from a Device, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.

### Launch edge

Edge of the clock at which the source register launches the signal onto the line.

### Latch edge

Edge of the clock at which the destination register latches in the signal.

## Driver Features

The SPI master driver governs communications of Hosts with Devices. The driver supports the following features:

- Multi-threaded environments
- Transparent handling of DMA transfers while reading and writing data
- Automatic time-division multiplexing of data coming from different Devices on the same signal bus, see SPI.

### Warning:

The SPI master driver has the concept of multiple Devices connected to a single bus (sharing a single ESP32 SPI peripheral). As long as each Device is accessed by only one task, the driver is thread safe. However, if multiple tasks try to access the same SPI Device, the driver is not thread-safe. In this case, it is recommended to either:

- Refactor your application so that each SPI peripheral is only accessed by a single task at a time.
- Add a mutex lock around the shared Device using `xSemaphoreCreateMutex`.

## SPI 特性

Espressif Systems 1233 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
SPI 主机

SPI 总线锁 为了多路复用来自不同驱动的不同设备，包括 SPI 主机、SPI Flash 等驱动，每个 SPI 总线上都只有一个 SPI 总线锁。驱动程序可以通过对锁的仲裁，将设备连接到总线上。

每个总线锁都已经初始化并注册了后台服务（BG）。所有请求在总线上进行传输的设备都应等到 BG 被成功禁用后再开始传输。

- 在 SPI 总线上，BG 为高速缓存。总线锁可以在设备操作开始前禁用高速缓存，并在设备释放锁后再次启用它，SPI 上的任何设备都无法使用 ISR，因为当高速缓存被禁用时，让出当前任务的执行权是没有意义的。
- 在 SPI 总线上使用 SPI 主机驱动程序时，存在一些限制。请参见 Notes on Using the SPI Master driver on SPI Bus。
- 对于其他总线，驱动程序可以将其 ISR 注册为 BG。当一个设备任务要求独占总线时，总线锁将阻塞该任务，同时禁用 ISR，并在 ISR 被成功禁用后，解除对该任务的阻塞。当任务释放锁时，如果 ISR 中还有待处理的事务，锁也将尝试恢复 ISR。

SPI Transactions

An SPI bus transaction consists of five phases which can be found in the table below. Any of these phases can be skipped.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>In this phase, a command (0-16 bit) is written to the bus by the Host.</td>
</tr>
<tr>
<td>Address</td>
<td>In this phase, an address (0-64 bit) is transmitted over the bus by the Host.</td>
</tr>
<tr>
<td>Dummy</td>
<td>This phase is configurable and is used to meet the timing requirements.</td>
</tr>
<tr>
<td>Write</td>
<td>Host sends data to a Device. This data follows the optional command and address phases and is indistinguishable from them at the electrical level.</td>
</tr>
<tr>
<td>Read</td>
<td>Device sends data to its Host.</td>
</tr>
</tbody>
</table>

The attributes of a transaction are determined by the bus configuration structure `spi_bus_config_t`, device configuration structure `spi_device_interface_config_t`, and transaction configuration structure `spi_transaction_t`.

An SPI Host can send full-duplex transactions, during which the read and write phases occur simultaneously. The total transaction length is determined by the sum of the following members:

- `spi_device_interface_config_t::command_bits`
- `spi_device_interface_config_t::address_bits`
- `spi_transaction_t::length`

While the member `spi_transaction_t::rxlength` only determines the length of data received into the buffer.

In half-duplex transactions, the read and write phases are not simultaneous (one direction at a time). The lengths of the write and read phases are determined by `spi_transaction_t::length` and `spi_transaction_t::rxlength` respectively.

The command and address phases are optional, as not every SPI device requires a command and/or address. This is reflected in the Device’s configuration: if `spi_device_interface_config_t::command_bits` and/or `spi_device_interface_config_t::address_bits` are set to zero, no command or address phase will occur.

The read and write phases can also be optional, as not every transaction requires both writing and reading data. If `spi_transaction_t::rx_buffer` is NULL and `SPI_TRANS_USE_RXDATA` is not set, the read phase is skipped. If `spi_transaction_t::tx_buffer` is NULL and `SPI_TRANS_USE_TXDATA` is not set, the write phase is skipped.
The driver supports two types of transactions: the interrupt transactions and polling transactions. The programmer can choose to use a different transaction type per Device. If your Device requires both transaction types, see Notes on Sending Mixed Transactions to the Same Device.

**Interrupt Transactions** Interrupt transactions will block the transaction routine until the transaction completes, thus allowing the CPU to run other tasks.

An application task can queue multiple transactions, and the driver will automatically handle them one-by-one in the interrupt service routine (ISR). It allows the task to switch to other procedures until all the transactions complete.

**Polling Transactions** Polling transactions do not use interrupts. The routine keeps polling the SPI Host’s status bit until the transaction is finished.

All the tasks that use interrupt transactions can be blocked by the queue. At this point, they will need to wait for the ISR to run twice before the transaction is finished. Polling transactions save time otherwise spent on queue handling and context switching, which results in smaller transaction duration. The disadvantage is that the CPU is busy while these transactions are in progress.

The `spi_device_polling_end()` routine needs an overhead of at least 1 us to unblock other tasks when the transaction is finished. It is strongly recommended to wrap a series of polling transactions using the functions `spi_device_acquire_bus()` and `spi_device_release_bus()` to avoid the overhead. For more information, see Bus Acquiring.

**Transaction Line Mode** Supported line modes for ESP32 are listed as follows, to make use of these modes, set the member `flags` in the struct `spi_transaction_t` as shown in the Transaction Flag column. If you want to check if corresponding IO pins are set or not, set the member `flags` in the `spi_bus_config_t` as shown in the Bus IO setting Flag column.

<table>
<thead>
<tr>
<th>Mode name</th>
<th>Command Line Width</th>
<th>Address Line Width</th>
<th>Data Line Width</th>
<th>Transaction Flag</th>
<th>Bus IO setting Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal SPI</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dual Output</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>SPI_TRANS_MODE_DIO</td>
<td>SPICOM-MON_BUSFLAG_DUAL</td>
</tr>
<tr>
<td>Dual I/O</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>SPI_TRANS_MODE_DIO SPI_TRANS_MULTILINE_ADDR</td>
<td></td>
</tr>
<tr>
<td>Quad Output</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>SPI_TRANS_MODE_QIO</td>
<td>SPICOM-MON_BUSFLAG_QUAD</td>
</tr>
<tr>
<td>Quad I/O</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>SPI_TRANS_MODE_QIO SPI_TRANS_MULTILINE_ADDR</td>
<td></td>
</tr>
</tbody>
</table>

**Command and Address Phases** During the command and address phases, the members `spi_transaction_t::cmd` and `spi_transaction_t::addr` are sent to the bus, nothing is read at this time. The default lengths of the command and address phases are set in `spi_device_interface_config_t` by calling `spi_bus_add_device()`. If the flags `SPI_TRANS_VARIABLE_CMD` and `SPI_TRANS_VARIABLE_ADDR` in the member `spi_transaction_t::flags` are not set, the driver automatically sets the length of these phases to default values during Device initialization.

If the lengths of the command and address phases need to be variable, declare the struct `spi_transaction_ext_t`, set the flags `SPI_TRANS_VARIABLE_CMD` and/or `SPI_TRANS_VARIABLE_ADDR` in the member `spi_transaction_ext_t::base` and configure the rest of base as usual. Then the length of each phase will be equal to `spi_transaction_ext_t::command_bits` and `spi_transaction_ext_t::address_bits` set in the struct `spi_transaction_ext_t`.

Submit Document Feedback
If the command and address phase need to be as the same number of lines as data phase, you need to set `SPI_TRANS_MULTILINE_CMD` and/or `SPI_TRANS_MULTILINE_ADDR` to the `flags` member in the struct `spi_transaction_t`. Also see `Transaction Line Mode`

**Write and Read Phases** Normally, the data that needs to be transferred to or from a Device will be read from or written to a chunk of memory indicated by the members `spi_transaction_t::rx_buffer` and `spi_transaction_t::tx_buffer`. If DMA is enabled for transfers, the buffers are required to be:

1. Allocated in DMA-capable internal memory. If `external PSRAM is enabled`, this means using `pvPortMallocCaps(size, MALLOC_CAP_DMA)`.
2. 32-bit aligned (staring from a 32-bit boundary and having a length of multiples of 4 bytes).

If these requirements are not satisfied, the transaction efficiency will be affected due to the allocation and copying of temporary buffers.

If using more than one data lines to transmit, please set `SPI_DEVICE_HALFDUPLex` flag for the member `flags` in the struct `spi_device_interface_config_t`. And the member `flags` in the struct `spi_transaction_t` should be set as described in `Transaction Line Mode`.

备注: Half-duplex transactions with both read and write phases are not supported when using DMA. For details and workarounds, see Known Issues.

**Bus Acquiring** Sometimes you might want to send SPI transactions exclusively and continuously so that it takes as little time as possible. For this, you can use bus acquiring, which helps to suspend transactions (both polling or interrupt) to other devices until the bus is released. To acquire and release a bus, use the functions `spi_device_acquire_bus()` and `spi_device_release_bus()`.

**Driver Usage**

- Initialize an SPI bus by calling the function `spi_bus_initialize()`. Make sure to set the correct I/O pins in the struct `spi_bus_config_t`. Set the signals that are not needed to -1.
- Register a Device connected to the bus with the driver by calling the function `spi_bus_add_device()`. Make sure to configure any timing requirements the device might need with the parameter `dev_config`. You should now have obtained the Device’s handle which will be used when sending a transaction to it.
- To interact with the Device, fill one or more `spi_transaction_t` structs with any transaction parameters required. Then send the structs either using a polling transaction or an interrupt transaction:
  - **Interrupt** Either queue all transactions by calling the function `spi_device_queue_trans()` and, at a later time, query the result using the function `spi_device_get_trans_result()`, or handle all requests synchronously by feeding them into `spi_device_transmit()`.
  - **Polling** Call the function `spi_device_polling_transmit()` to send polling transactions. Alternatively, if you want to insert something in between, send the transactions by using `spi_device_polling_start()` and `spi_device_polling_end()`.
- (Optional) To perform back-to-back transactions with a Device, call the function `spi_device_acquire_bus()` before sending transactions and `spi_device_release_bus()` after the transactions have been sent.
- (Optional) To unload the driver for a certain Device, call `spi_bus_remove_device()` with the Device handle as an argument.
- (Optional) To remove the driver for a bus, make sure no more drivers are attached and call `spi_bus_free()`.

The example code for the SPI master driver can be found in the `peripherals/spi_master` directory of ESP-IDF examples.

**Transactions with Data Not Exceeding 32 Bits** When the transaction data size is equal to or less than 32 bits, it will be sub-optimal to allocate a buffer for the data. The data can be directly stored in the transaction struct instead. For transmitted data, it can be achieved by using the
spi_transaction_t::tx_data member and setting the SPI_TRANS_USE_TXDATA flag on the transmission. For received data, use spi_transaction_t::rx_data and set SPI_TRANS_USE_RXDATA. In both cases, do not touch the spi_transaction_t::tx_buffer or spi_transaction_t::rx_buffer members, because they use the same memory locations as spi_transaction_t::tx_data and spi_transaction_t::rx_data.

Transactions with Integers Other Than uint8_t  An SPI Host reads and writes data into memory byte by byte. By default, data is sent with the most significant bit (MSB) first, as LSB first used in rare cases. If a value less than 8 bits needs to be sent, the bits should be written into memory in the MSB first manner.

For example, if 0b00010 needs to be sent, it should be written into a uint8_t variable, and the length for reading should be set to 5 bits. The Device will still receive 8 bits with 3 additional “random” bits, so the reading must be performed correctly.

On top of that, ESP32 is a little-endian chip, which means that the least significant byte of uint16_t and uint32_t variables is stored at the smallest address. Hence, if uint16_t is stored in memory, bits [7:0] are sent first, followed by bits [15:8].

For cases when the data to be transmitted has the size differing from uint8_t arrays, the following macros can be used to transform data to the format that can be sent by the SPI driver directly:

- SPI_SWAP_DATA_TX for data to be transmitted
- SPI_SWAP_DATA_RX for data received

Notes on Sending Mixed Transactions to the Same Device  To reduce coding complexity, send only one type of transactions (interrupt or polling) to one Device. However, you still can send both interrupt and polling transactions alternately. The notes below explain how to do this.

The polling transactions should be initiated only after all the polling and interrupt transactions are finished.

Since an unfinished polling transaction blocks other transactions, please do not forget to call the function spi_device_polling_end() after spi_device_polling_start() to allow other transactions or to allow other Devices to use the bus. Remember that if there is no need to switch to other tasks during your polling transaction, you can initiate a transaction with spi_device_polling_transmit() so that it will be ended automatically.

In-flight polling transactions are disturbed by the ISR operation to accommodate interrupt transactions. Always make sure that all the interrupt transactions sent to the ISR are finished before you call spi_device_polling_start(). To do that, you can keep calling spi_device_get_trans_result() until all the transactions are returned.

To have better control of the calling sequence of functions, send mixed transactions to the same Device only within a single task.

Notes on Using the SPI Master driver on SPI1 Bus  Though the SPIMaster feature makes it possible to use SPI Master driver on the SPI1 bus, it’s still tricky and needs a lot of special treatment. It’s a feature for advanced developers.

To use SPI Master driver on SPI1 bus, you have to take care of two problems:

1. The code and data, required at the meanwhile the driver is operating SPI1 bus, should be in the internal memory. SPI1 bus is shared among devices and the cache for data (code) in the Flash as well as the PSRAM. The cache should be disabled during the other drivers are operating the SPI1 bus. Hence the data (code) in the flash as well as the PSRAM cannot be fetched at the meanwhile the driver acquires the SPI1 bus by:
   - Explicit bus acquiring between spi_device_acquire_bus() and spi_device_release_bus()
   - Implicit bus acquiring between spi_device_polling_start() and spi_device_polling_end() (or inside spi_device_polling_transmit())
During the time above, all other tasks and most ISRs will be disabled (see IRAM 安全中断处理程序). Application code and data used by current task should be placed in internal memory (DRAM or IRAM), or already in the ROM. Access to external memory (flash code, const data in the flash, and static/heap data in the PSRAM) will cause a Cache disabled but cached memory region accessed exception. For differences between IRAM, DRAM, and flash cache, please refer to the application memory layout documentation.

To place functions into the IRAM, you can either:
1. Add `IRAM_ATTR` (include esp_attr.h) to the function like:
   ```c
   IRAM_ATTR void foo(void) {}
   ```
   Please note that when a function is inlined, it will follow its caller’s segment, and the attribute will not take effect. You may need to use `NOLINE_ATTR` to avoid this.
2. Use the `noflash` placement in the linker.lf. See more in 链接器脚本生成机制. Please note that, some code may be transformed into lookup table in the const data by the compiler, so `noflash_text` is not safe. Please do take care that the optimization level may affect the compiler behavior of inline, or transforming some code into lookup table in the const data, etc.

To place data into the DRAM, you can either:
1. Add `DRAM_ATTR` (include esp_attr.h) to the data definition like:
   ```c
   DRAM_ATTR int g_foo = 3;
   ```
2. Use the `noflash` placement in the linker.lf. See more in 链接器脚本生成机制.

Please also see the example peripherals/spi_master/hd_eeprom.

**GPIO Matrix and IO_MUX** Most of ESP32’s peripheral signals have direct connection to their dedicated IO_MUX pins. However, the signals can also be routed to any other available pins using the less direct GPIO matrix. If at least one signal is routed through the GPIO matrix, then all signals will be routed through it.

The GPIO matrix introduces flexibility of routing but also brings the following disadvantages:

- Increases the input delay of the MISO signal, which makes MISO setup time violations more likely. If SPI needs to operate at high speeds, use dedicated IO_MUX pins.
- Allows signals with clock frequencies only up to 40 MHz, as opposed to 80 MHz if IO_MUX pins are used.

**备注：** For more details about the influence of the MISO input delay on the maximum clock frequency, see Timing Considerations.

The IO_MUX pins for SPI buses are given below.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>SPI2</th>
<th>SPI3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS0*</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>SCLK</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>MISO</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>MOSI</td>
<td>13</td>
<td>23</td>
</tr>
<tr>
<td>QUADWP</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>QUADHD</td>
<td>4</td>
<td>21</td>
</tr>
</tbody>
</table>

- Only the first Device attached to the bus can use the CS0 pin.

**Transfer Speed Considerations**

There are three factors limiting the transfer speed:

- Transaction interval
- SPI clock frequency
- Cache miss of SPI functions, including callbacks

The main parameter that determines the transfer speed for large transactions is clock frequency. For multiple small transactions, the transfer speed is mostly determined by the length of transaction intervals.
Transaction Duration  
Transaction duration includes setting up SPI peripheral registers, copying data to FIFOs or setting up DMA links, and the time for SPI transaction.

Interrupt transactions allow appending extra overhead to accommodate the cost of FreeRTOS queues and the time needed for switching between tasks and the ISR.

For interrupt transactions, the CPU can switch to other tasks when a transaction is in progress. This saves the CPU time but increases the transaction duration. See Interrupt Transactions. For polling transactions, it does not block the task but allows to do polling when the transaction is in progress. For more information, see Polling Transactions.

If DMA is enabled, setting up the linked list requires about 2 us per transaction. When a master is transferring data, it automatically reads the data from the linked list. If DMA is not enabled, the CPU has to write and read each byte from the FIFO by itself. Usually, this is faster than 2 us, but the transaction length is limited to 64 bytes for both write and read.

Typical transaction duration for one byte of data are given below.

- Interrupt Transaction via DMA: 28 µs.
- Interrupt Transaction via CPU: 25 µs.
- Polling Transaction via DMA: 10 µs.
- Polling Transaction via CPU: 8 µs.

SPI Clock Frequency  
The driver support setting an SPI peripheral to different clock frequencies. Actual clock frequency may not be exactly equal to the number you set, it will be re-calculated by the driver to the nearest hardware compatible number, you can call `spi_device_get_actual_freq()` to get the actual frequency computed by driver.

Theoretical maximum transfers speed of Write or Read phase can be calculated according to the table below:

<table>
<thead>
<tr>
<th>Line Width of Write/Read phase</th>
<th>Speed (Bps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Line</td>
<td>SPI Frequency / 8</td>
</tr>
<tr>
<td>2-Line</td>
<td>SPI Frequency / 4</td>
</tr>
<tr>
<td>4-Line</td>
<td>SPI Frequency / 2</td>
</tr>
</tbody>
</table>

The transfer speed calculation of other phases (command, address, dummy) are similar.

If the clock frequency is too high, the use of some functions might be limited. See Timing Considerations.

Cache Miss  
The default config puts only the ISR into the IRAM. Other SPI related functions, including the driver itself and the callback, might suffer from cache misses and will need to wait until the code is read from flash. Select `CONFIG_SPI_MASTER_IN_IRAM` to put the whole SPI driver into IRAM and put the entire callback(s) and its callee functions into IRAM to prevent cache misses.

For an interrupt transaction, the overall cost is \(20+8n/F_{spi}[MHz]\) [us] for n bytes transferred in one transaction. Hence, the transferring speed is: \(n/(20+8n/F_{spi})\). An example of transferring speed at 8 MHz clock speed is given in the following table:

<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Transaction Interval (us)</th>
<th>Transaction Length (bytes)</th>
<th>Total Time (us)</th>
<th>Total Speed (KBps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>25</td>
<td>1</td>
<td>26</td>
<td>38.5</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>8</td>
<td>33</td>
<td>242.4</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>16</td>
<td>41</td>
<td>490.2</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>64</td>
<td>89</td>
<td>719.1</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>128</td>
<td>153</td>
<td>836.6</td>
</tr>
</tbody>
</table>

When a transaction length is short, the cost of transaction interval is high. If possible, try to squash several short transactions into one transaction to achieve a higher transfer speed.

Please note that the ISR is disabled during flash operation by default. To keep sending transactions during flash operations, enable `CONFIG_SPI_MASTER_ISR_IN_IRAM` and set `ESP_INTR_FLAG_IRAM` in the member
spi_bus_config_t::intr_flags. In this case, all the transactions queued before starting flash operations will be handled by the ISR in parallel. Also note that the callback of each Device and their callee functions should be in IRAM, or your callback will crash due to cache miss. For more details, see IRAM 安全中断处理程序.

Timing Considerations

As shown in the figure below, there is a delay on the MISO line after the SCLK launch edge and before the signal is latched by the internal register. As a result, the MISO pin setup time is the limiting factor for the SPI clock speed. When the delay is too long, the setup slack is < 0, which means the setup timing requirement is violated and the reading might be incorrect.

![Timing Diagram](image)

The maximum allowed frequency is dependent on:

- spi_device_interface_config_t::input_delay_ns - maximum data valid time on the MISO bus after a clock cycle on SCLK starts
- If the IO_MUX pin or the GPIO Matrix is used

When the GPIO matrix is used, the maximum allowed frequency is reduced to about 33~77% in comparison to the existing input delay. To retain a higher frequency, you have to use the IO_MUX pins or the dummy bit workaround. You can obtain the maximum reading frequency of the master by using the function spi_get_freq_limit().

**Dummy bit workaround:** Dummy clocks, during which the Host does not read data, can be inserted before the read phase begins. The Device still sees the dummy clocks and sends out data, but the Host does not read until the read phase comes. This compensates for the lack of the MISO setup time required by the Host and allows the Host to do reading at a higher frequency.

In the ideal case, if the Device is so fast that the input delay is shorter than an APB clock cycle - 12.5 ns - the maximum frequency at which the Host can read (or read and write) in different conditions is as follows:

<table>
<thead>
<tr>
<th>Frequency Limit (MHz)</th>
<th>Dummy Bits Used By Driver</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO matrix</td>
<td>IO_MUX pins</td>
<td></td>
</tr>
<tr>
<td>26.6</td>
<td>80</td>
<td>No</td>
</tr>
<tr>
<td>40</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>

If the Host only writes data, the dummy bit workaround and the frequency check can be disabled by setting the bit SPI_DEVICE_NO_DUMMY in the member spi_device_interface_config_t::flags. When disabled, the output frequency can be 80MHz, even if the GPIO matrix is used.
spi_device_interface_config_t::flags

The SPI master driver still works even if the `spi_device_interface_config_t::input_delay_ns` in the structure `spi_device_interface_config_t` is set to 0. However, setting an accurate value helps to:

- Calculate the frequency limit for full-duplex transactions
- Compensate the timing correctly with dummy bits for half-duplex transactions

You can approximate the maximum data valid time after the launch edge of SPI clocks by checking the statistics in the AC characteristics chapter of your Device’s specification or measure the time using an oscilloscope or logic analyzer.

Please note that the actual PCB layout design and excessive loads may increase the input delay. It means that non-optimal wiring and/or a load capacitor on the bus will most likely lead to input delay values exceeding the values given in the Device specification or measured while the bus is floating.

Some typical delay values are shown in the following table. (These data are retrieved when the slave device is on a different physical chip)

<table>
<thead>
<tr>
<th>Device</th>
<th>Input delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal Device</td>
<td>0</td>
</tr>
<tr>
<td>ESP32 slave using IO_MUX*</td>
<td>50</td>
</tr>
<tr>
<td>ESP32 slave using GPIO_MUX*</td>
<td>75</td>
</tr>
</tbody>
</table>

The MISO path delay (valid time) consists of a slave’s input delay plus master’s GPIO matrix delay. This delay determines the frequency limit above which full-duplex transfers will not work as well as the dummy bits used in the half-duplex transactions. The frequency limit is:

\[
Freq \ limit \ [MHz] = \frac{80}{floor(MISO \ delay[ns]/12.5) + 1}
\]

The figure below shows the relationship between frequency limit and input delay. Two extra APB clock cycle periods should be added to the MISO delay if the master uses the GPIO matrix.

Corresponding frequency limits for different Devices with different input delay times are shown in the table below.
<table>
<thead>
<tr>
<th>Master</th>
<th>Input delay (ns)</th>
<th>MISO path delay (ns)</th>
<th>Freq. limit (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_MUX (0ns)</td>
<td>0</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>50</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>75</td>
<td>11.43</td>
</tr>
<tr>
<td>GPIO (25ns)</td>
<td>0</td>
<td>25</td>
<td>26.67</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>75</td>
<td>11.43</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>100</td>
<td>8.89</td>
</tr>
</tbody>
</table>

**Known Issues**

1. Half-duplex transactions are not compatible with DMA when both writing and reading phases are used. If such transactions are required, you have to use one of the alternative solutions:
   1. Use full-duplex transactions instead.
   2. **Disable DMA by setting the bus initialization function’s last parameter to 0 as follows:**
      ```c
      ret=spi_bus_initialize(VSPI_HOST, &buscfg, 0);
      ```
      This can prohibit you from transmitting and receiving data longer than 64 bytes.
   3. Try using the command and address fields to replace the write phase.

2. Full-duplex transactions are not compatible with the *dummy bit workaround*, hence the frequency is limited. See [dummy bit speed-up workaround](#).

3. *dummy_bits* in `spi_device_interface_config_t` and `spi_transaction_ext_t` are not available when SPI read and write phases are both enabled (regardless of full duplex or half duplex mode).
4. `cs_ena_pretrans` is not compatible with the command and address phases of full-duplex transactions.

**Application Example**

The code example for using the SPI master half duplex mode to read/write a AT93C46D EEPROM (8-bit mode) can be found in the `peripherals/spi_master/hd_eeprom` directory of ESP-IDF examples.

**API Reference - SPI Common**

**Header File**

- `components/hal/include/hal/spi_types.h`

**Structures**

struct `spi_line_mode_t`

Line mode of SPI transaction phases: CMD, ADDR, DOUT/DIN.

**Public Members**

- `uint8_t cmd_lines`
  The line width of command phase, e.g. 2-line-cmd-phase.

- `uint8_t addr_lines`
  The line width of address phase, e.g. 1-line-addr-phase.

- `uint8_t data_lines`
  The line width of data phase, e.g. 4-line-data-phase.
Chapter 2. API

Enumerations

enum spi_host_device_t
Enum with the three SPI peripherals that are software-accessible in it.

Values:

enumerator SPI1_HOST
SPI1.

enumerator SPI2_HOST
SPI2.

enumerator SPI3_HOST
SPI3.

enumerator SPI_HOST_MAX
invalid host value

enum spi_clock_source_t
Values:

enumerator SPI_CLK_APB
Select APB as the source clock.

enumerator SPI_CLK_XTAL
Select XTAL as the source clock.

enum spi_event_t
SPI Events.

Values:

enumerator SPI_EV_BUF_TX
The buffer has sent data to master.

enumerator SPI_EV_BUF_RX
The buffer has received data from master.

enumerator SPI_EV_SEND_DMA_READY
Slave has loaded its TX data buffer to the hardware (DMA).

enumerator SPI EV_SEND
Master has received certain number of the data, the number is determined by Master.

enumerator SPI_EV_RECV_DMA_READY
Slave has loaded its RX data buffer to the hardware (DMA).

enumerator SPI EV_RECV
Slave has received certain number of data from master, the number is determined by Master.
Chapter 2. API 参考

enumerator SPI_EV_CMD9
   Received CMD9 from master.

enumerator SPI_EV_CMDA
   Received CMDA from master.

enumerator SPI_EV_TRANS
   A transaction has done.

enum spi_command_t
   SPI command.
   Values:
   enumerator SPI_CMD_HD_WRBUF
   enumerator SPI_CMD_HD_RDBUF
   enumerator SPI_CMD_HD_WRDMA
   enumerator SPI_CMD_HD_RDDMA
   enumerator SPI_CMD_HD_SEG_END
   enumerator SPI_CMD_HD_EN_QPI
   enumerator SPI_CMD_HD_WR_END
   enumerator SPI_CMD_HD_INT0
   enumerator SPI_CMD_HD_INT1
   enumerator SPI_CMD_HD_INT2

Header File
   • components/driver/include/driver/spi_common.h

Functions
esp_err_t spi_bus_initialize (spi_host_device_t host_id, const spi_bus_config_t *bus_config,
   spi_dma_chan_t dma_chan)
   Initialize a SPI bus.

警告： SPI0/1 is not supported

警告： If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.
The ISR of SPI is always executed on the core which calls this function. Never starve the ISR on this core or the SPI transactions will not be handled.

**Parameters**

- `host_id` – SPI peripheral that controls this bus
- `bus_config` – Pointer to a `spi_bus_config_t` struct specifying how the host should be initialized
- `dma_chan` – Selecting a DMA channel for an SPI bus allows transactions on the bus with size only limited by the amount of internal memory.
  - Selecting `SPI_DMA_DISABLED` limits the size of transactions.
  - Set to `SPI_DMA_DISABLED` if only the SPI flash uses this bus.
  - Set to `SPI_DMA_CH_AUTO` to let the driver to allocate the DMA channel.

**Return**

- `ESP_ERR_INVALID_ARG` if configuration is invalid
- `ESP_ERR_INVALID_STATE` if host already is in use
- `ESP_ERR_NOT_FOUND` if there is no available DMA channel
- `ESP_ERR_NO_MEM` if out of memory
- `ESP_OK` on success

```c
esp_err_t spi_bus_free(spi_host_device_t host_id)
```

Free a SPI bus.

In order for this to succeed, all devices have to be removed first.

**Parameters**

- `host_id` – SPI peripheral to free

**Return**

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_ERR_INVALID_STATE` if bus hasn’t been initialized before, or not all devices on the bus are freed
- `ESP_OK` on success

**Structures**

```c
struct spi_bus_config_t
```

This is a configuration structure for a SPI bus.

You can use this structure to specify the GPIO pins of the bus. Normally, the driver will use the GPIO matrix to route the signals. An exception is made when all signals either can be routed through the IO_MUX or are -1. In that case, the IO_MUX is used, allowing for >40MHz speeds.

**备注**：Be advised that the slave driver does not use the quadwp/quadhd lines and fields in `spi_bus_config_t` refering to these lines will be ignored and can thus safely be left uninitialized.

**Public Members**

```c
int mosi_io_num
```

GPIO pin for Master Out Slave In (=spi_d) signal, or -1 if not used.

```c
int data0_io_num
```

GPIO pin for spi data0 signal in quad/octal mode, or -1 if not used.
int miso_io_num
GPIO pin for Master In Slave Out (spi_q) signal, or -1 if not used.

int data1_io_num
GPIO pin for spi data1 signal in quad/octet mode, or -1 if not used.

int sclk_io_num
GPIO pin for SPI Clock signal, or -1 if not used.

int quadwp_io_num
GPIO pin for WP (Write Protect) signal, or -1 if not used.

int data2_io_num
GPIO pin for spi data2 signal in quad/octet mode, or -1 if not used.

int quadhd_io_num
GPIO pin for HD (Hold) signal, or -1 if not used.

int data3_io_num
GPIO pin for spi data3 signal in quad/octet mode, or -1 if not used.

int data4_io_num
GPIO pin for spi data4 signal in octal mode, or -1 if not used.

int data5_io_num
GPIO pin for spi data5 signal in octal mode, or -1 if not used.

int data6_io_num
GPIO pin for spi data6 signal in octal mode, or -1 if not used.

int data7_io_num
GPIO pin for spi data7 signal in octal mode, or -1 if not used.

int max_transfer_sz
Maximum transfer size, in bytes. Defaults to 4092 if 0 when DMA enabled, or to
SOC_SPI_MAXIMUM_BUFFER_SIZE if DMA is disabled.

uint32_t flags
Abilities of bus to be checked by the driver. Or-ed value of SPICOMMON_BUSFLAG_* flags.

int intr_flags
Interrupt flag for the bus to set the priority, and IRAM attribute, see esp_intr_alloc.h. Note that
the EDGE, INTRDISABLED attribute are ignored by the driver. Note that if ESP_INTR_FLAG_IRAM
is set, ALL the callbacks of the driver, and their callee functions, should be put in the IRAM.

 Macros

SPI_MAX_DMA_LEN
**SPI_SWAP_DATA_TX** (DATA, LEN)
Transform unsigned integer of length <= 32 bits to the format which can be sent by the SPI driver directly.

E.g. to send 9 bits of data, you can:

```c
uint16_t data = SPI_SWAP_DATA_TX(0x145, 9);
```

Then points `tx_buffer` to `&data`.

- **DATA** - Data to be sent, can be `uint8_t`, `uint16_t` or `uint32_t`.
- **LEN** - Length of data to be sent, since the SPI peripheral sends from the MSB, this helps to shift the data to the MSB.

**SPI_SWAP_DATA_RX** (DATA, LEN)
Transform received data of length <= 32 bits to the format of an unsigned integer.

E.g. to transform the data of 15 bits placed in a 4-byte array to integer:

```c
uint16_t data = SPI_SWAP_DATA_RX((uint32_t*)t->rx_data, 15);
```

- **DATA** - Data to be rearranged, can be `uint8_t`, `uint16_t` or `uint32_t`.
- **LEN** - Length of data received, since the SPI peripheral writes from the MSB, this helps to shift the data to the LSB.

**SPICOMMON_BUSFLAG_SLAVE**
Initialize I/O in slave mode.

**SPICOMMON_BUSFLAG_MASTER**
Initialize I/O in master mode.

**SPICOMMON_BUSFLAG_IOMUX_PINS**
Check using iomux pins. Or indicates the pins are configured through the IO mux rather than GPIO matrix.

**SPICOMMON_BUSFLAG_GPIO_PINS**
Force the signals to be routed through GPIO matrix. Or indicates the pins are routed through the GPIO matrix.

**SPICOMMON_BUSFLAG_SCLK**
Check existing of SCLK pin. Or indicates CLK line initialized.

**SPICOMMON_BUSFLAG_MISO**
Check existing of MISO pin. Or indicates MISO line initialized.

**SPICOMMON_BUSFLAG_MOSI**
Check existing of MOSI pin. Or indicates MOSI line initialized.

**SPICOMMON_BUSFLAG_DUAL**
Check MOSI and MISO pins can output. Or indicates bus able to work under DIO mode.

**SPICOMMON_BUSFLAG_WPHD**
Check existing of WP and HD pins. Or indicates WP & HD pins initialized.
**SPICOMMON_BUSFLAG_QUAD**
Check existing of MOSI/MISO/WP/HD pins as output. Or indicates bus able to work under QIO mode.

**SPICOMMON_BUSFLAG_IO4_IO7**
Check existing of IO4~IO7 pins. Or indicates IO4~IO7 pins initialized.

**SPICOMMON_BUSFLAG_OCTAL**
Check existing of MOSI/MISO/HD/SPIIO4/SPIIO5/SPIIO6/SPIIO7 pins as output. Or indicates bus able to work under octal mode.

**SPICOMMON_BUSFLAG_NATIVE_PINS**

**Type Definitions**

typedef *spi_common_dma_t* spi_dma_chan_t

**Enumerations**
enum *spi_common_dma_t*
SPI DMA channels.

*Values:*

enumerator *SPI_DMA_DISABLED*
   Do not enable DMA for SPI.

enumerator *SPI_DMA_CH1*
   Enable DMA, select DMA Channel 1.

enumerator *SPI_DMA_CH2*
   Enable DMA, select DMA Channel 2.

enumerator *SPI_DMA_CH_AUTO*
   Enable DMA, channel is automatically selected by driver.

**API Reference - SPI Master**

**Header File**
- components/driver/include/driver/spi_master.h

**Functions**

`esp_err_t spi_bus_add_device (spi_host_device_t host_id, const spi_device_interface_config_t *dev_config, spi_device_handle_t *handle)`
Allocate a device on a SPI bus.

This initializes the internal structures for a device, plus allocates a CS pin on the indicated SPI master peripheral and routes it to the indicated GPIO. All SPI master devices have three CS pins and can thus control up to three devices.
Chapter 2. API

While in general, speeds up to 80MHz on the dedicated SPI pins and 40MHz on GPIO-matrix-routed pins are supported, full-duplex transfers routed over the GPIO matrix only support speeds up to 26MHz.

### Parameters
- **host_id** - SPI peripheral to allocate device on
- **dev_config** - SPI interface protocol config for the device
- **handle** - Pointer to variable to hold the device handle

### Returns
- **ESP_ERR_INVALID_ARG** if parameter is invalid or configuration combination is not supported (e.g. dev_config->post_cb isn’t set while flag `SPI_DEVICE_NO_RETURN_RESULT` is enabled)
- **ESP_ERR_NOT_FOUND** if host doesn’t have any free CS slots
- **ESP_ERR_NO_MEM** if out of memory
- **ESP_OK** on success

```c
esp_err_t spi_bus_remove_device(spi_device_handle_t handle)
```

Remove a device from the SPI bus.

- **handle** - Device handle to free

### Returns
- **ESP_ERR_INVALID_ARG** if parameter is invalid
- **ESP_ERR_INVALID_STATE** if device already is freed
- **ESP_OK** on success

```c
esp_err_t spi_device_queue_trans(spi_device_handle_t handle, spi_transaction_t *trans_desc, TickType_t ticks_to_wait)
```

Queue a SPI transaction for interrupt transaction execution. Get the result by `spi_device_get_trans_result`.

### Returns
- **ESP_ERR_INVALID_ARG** if parameter is invalid. This can happen if `SPI_TRANS_CS_KEEP_ACTIVE` flag is specified while the bus was not acquired (`spi_device_acquire_bus()` should be called first)
- **ESP_ERR_TIMEOUT** if there was no room in the queue before ticks_to_wait expired
- **ESP_ERR_NO_MEM** if allocating DMA-capable temporary buffer failed
- **ESP_ERR_INVALID_STATE** if previous transactions are not finished
- **ESP_OK** on success

```c
esp_err_t spi_device_get_trans_result(spi_device_handle_t handle, spi_transaction_t **trans_desc, TickType_t ticks_to_wait)
```

Get the result of a SPI transaction queued earlier by `spi_device_queue_trans`.

This routine will wait until a transaction to the given device successfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or re-use the buffers.

### Parameters
- **handle** - Device handle obtained using `spi_host_add_dev`
• **trans_desc** - Pointer to variable able to contain a pointer to the description of the transaction that is executed. The descriptor should not be modified until the descriptor is returned by `spi_device_get_trans_result`.
• **ticks_to_wait** - Ticks to wait until there’s a returned item; use `portMAX_DELAY` to never time out.

回复
• ESP_ERR_INVALID_ARG if parameter is invalid
• ESP_ERR_NOT_SUPPORTED if flag `SPI_DEVICE_NO_RETURN_RESULT` is set
• ESP_ERR_TIMEOUT if there was no completed transaction before `ticks_to_wait` expired
• ESP_OK on success

```c
esp_err_t spi_device_transmit(spi_device_handle_t handle, spi_transaction_t *trans_desc)
```

Send a SPI transaction, wait for it to complete, and return the result.

This function is the equivalent of calling `spi_device_queue_trans()` followed by `spi_device_get_trans_result()`. Do not use this when there is still a transaction separately queued (started) from `spi_device_queue_trans()` or polling_start/transmit that has’nt been finalized.

备注: This function is not thread safe when multiple tasks access the same SPI device. Normally a device cannot start (queue) polling and interrupt transactions simultaneously.

参数
• **handle** - Device handle obtained using `spi_host_add_dev`
• **trans_desc** - Description of transaction to execute

返回
• ESP_ERR_INVALID_ARG if parameter is invalid
• ESP_OK on success

```c
esp_err_t spi_device_polling_start(spi_device_handle_t handle, spi_transaction_t *trans_desc, TickType_t ticks_to_wait)
```

Immediately start a polling transaction.

备注: Normally a device cannot start (queue) polling and interrupt transactions simultaneously. Moreover, a device cannot start a new polling transaction if another polling transaction is not finished.

参数
• **handle** - Device handle obtained using `spi_host_add_dev`
• **trans_desc** - Description of transaction to execute
• **ticks_to_wait** - Ticks to wait until there’s room in the queue; currently only `portMAX_DELAY` is supported.

返回
• ESP_ERR_INVALID_ARG if parameter is invalid. This can happen if `SPI_TRANS_CS_KEEP_ACTIVE` flag is specified while the bus was not acquired (`spi_device_acquire_bus()` should be called first)
• ESP_ERR_TIMEOUT if the device cannot get control of the bus before `ticks_to_wait` expired
• ESP_ERR_NO_MEM if allocating DMA-capable temporary buffer failed
• ESP_ERR_INVALID_STATE if previous transactions are not finished
• ESP_OK on success

```c
esp_err_t spi_device_polling_end(spi_device_handle_t handle, TickType_t ticks_to_wait)
```

Poll until the polling transaction ends.

This routine will not return until the transaction to the given device has successfully completed. The task is not blocked, but actively busy-spins for the transaction to be completed.
### Chapter 2. API 参考

#### 参数
- **handle** - Device handle obtained using `spi_host_add_dev`
- **ticks_to_wait** - Ticks to wait until there’s a returned item; use `portMAX_DELAY` to never time out.

#### 返回
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if the transaction cannot finish before `ticks_to_wait` expired
- ESP_OK on success

```c
esp_err_t spi_device_polling_transmit (spi_device_handle_t handle, spi_transaction_t *trans_desc)
```

Send a polling transaction, wait for it to complete, and return the result.

This function is the equivalent of calling `spi_device_polling_start()` followed by `spi_device_polling_end()`. Do not use this when there is still a transaction that hasn’t been finalized.

**备注:** This function is not thread safe when multiple tasks access the same SPI device. Normally a device cannot start (queue) polling and interrupt transactions simultaneously.

#### 参数
- **handle** - Device handle obtained using `spi_host_add_dev`
- **trans_desc** - Description of transaction to execute

#### 返回
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

```c
esp_err_t spi_device_acquire_bus (spi_device_handle_t device, TickType_t wait)
```

Occupies the SPI bus for a device to do continuous transactions.

Transactions to all other devices will be put off until `spi_device_release_bus` is called.

**备注:** The function will wait until all the existing transactions have been sent.

#### 参数
- **device** - The device to occupy the bus.
- **wait** - Time to wait before the the bus is occupied by the device. Currently MUST set to `portMAX_DELAY`.

#### 返回
- ESP_ERR_INVALID_ARG: `wait` is not set to `portMAX_DELAY`.
- ESP_OK: Success.

```c
void spi_device_release_bus (spi_device_handle_t dev)
```

Releases the SPI bus occupied by the device. All other devices can start sending transactions.

**参数** dev – The device to release the bus.

```c
esp_err_t spi_device_get_actual_freq (spi_device_handle_t handle, int *freq_khz)
```

Calculates working frequency for specific device.

**参数**
- **handle** - SPI device handle
- **freq_khz** - [out] output parameter to hold calculated frequency in kHz

#### 返回
- ESP_ERR_INVALID_ARG: `handle` or `freq_khz` parameter is NULL
- ESP_OK: Success

```c
int spi_get_actual_clock (int fapb, int hz, int duty_cycle)
```

Calculates the working frequency that is most close to desired frequency.
The frequency of the APB clock should be `APB_CLK_FREQ`.

Desired working frequency

Duty cycle of the SPI clock

Actual working frequency that most fit.

```c
void spi_get_timing(bool gpio_is_used, int input_delay_ns, int eff_clk, int *dummy_o, int *cycles_remain_o)
```

Calculate the timing settings of specified frequency and settings.

**备注:** If `*dummy_o` is not zero, it means dummy bits should be applied in half duplex mode, and full duplex mode may not work.

```c
int spi_get_freq_limit(bool gpio_is_used, int input_delay_ns)
```

Get the frequency limit of current configurations. SPI master working at this limit is OK, while above the limit, full duplex mode and DMA will not work, and dummy bits will be applied in the half duplex mode.

```c
struct spi_device_interface_config_t
```

This is a configuration for a SPI slave device that is connected to one of the SPI buses.

**Public Members**

```c
uint8_t command_bits
```

Default amount of bits in command phase (0-16), used when `SPI_TRANS_VARIABLE_CMD` is not used, otherwise ignored.

```c
uint8_t address_bits
```

Default amount of bits in address phase (0-64), used when `SPI_TRANS_VARIABLE_ADDR` is not used, otherwise ignored.

```c
uint8_t dummy_bits
```

Amount of dummy bits to insert between address and data phase.

```c
uint8_t mode
```

SPI mode, representing a pair of (CPOL, CPHA) configuration:

- 0: (0, 0)
Chapter 2. API

- 1: (0, 1)
- 2: (1, 0)
- 3: (1, 1)

uint16_t duty_cycle_pos
Duty cycle of positive clock, in 1/256th increments (128 = 50%/50% duty). Setting this to 0 (=not setting it) is equivalent to setting this to 128.

uint16_t cs_ena_pretrans
Amount of SPI bit-cycles the cs should be activated before the transmission (0-16). This only works on half-duplex transactions.

uint8_t cs_ena_posttrans
Amount of SPI bit-cycles the cs should stay active after the transmission (0-16)

int clock_speed_hz
Clock speed, divisors of 80MHz, in Hz. See SPI_MASTER_FREQ_*.

int input_delay_ns
Maximum data valid time of slave. The time required between SCLK and MISO valid, including the possible clock delay from slave to master. The driver uses this value to give an extra delay before the MISO is ready on the line. Leave at 0 unless you know you need a delay. For better timing performance at high frequency (over 8MHz), it’s suggest to have the right value.

int spics_io_num
CS GPIO pin for this device, or -1 if not used.

uint32_t flags
Bitwise OR of SPI_DEVICE_* flags.

int queue_size
Transaction queue size. This sets how many transactions can be ‘in the air’ (queued using spi_device_queue_trans but not yet finished using spi_device_get_trans_result) at the same time.

transaction_cb_t pre_cb
Callback to be called before a transmission is started.

This callback is called within interrupt context should be in IRAM for best performance, see “Transferring Speed” section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

transaction_cb_t post_cb
Callback to be called after a transmission has completed.

This callback is called within interrupt context should be in IRAM for best performance, see “Transferring Speed” section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

struct spi_transaction_t
This structure describes one SPI transaction. The descriptor should not be modified until the transaction finishes.
Public Members

uint32_t flags
    Bitwise OR of SPI_TRANS_* flags.

uint16_t cmd
    Command data, of which the length is set in the command_bits of spi_device_interface_config_t.
    NOTE: this field, used to be "command" in ESP-IDF 2.1 and before, is re-written to be used in a new way in ESP-IDF 3.0.
    Example: write 0x0123 and command_bits=12 to send command 0x12, 0x3_ (in previous version, you may have to write 0x3_12).

uint64_t addr
    Address data, of which the length is set in the address_bits of spi_device_interface_config_t.
    NOTE: this field, used to be "address" in ESP-IDF 2.1 and before, is re-written to be used in a new way in ESP-IDF 3.0.
    Example: write 0x123400 and address_bits=24 to send address of 0x12, 0x34, 0x00 (in previous version, you may have to write 0x12340000).

size_t length
    Total data length, in bits.

size_t rxlength
    Total data length received, should be not greater than length in full-duplex mode (0 defaults this to the value of length).

void *user
    User-defined variable. Can be used to store eg transaction ID.

const void *tx_buffer
    Pointer to transmit buffer, or NULL for no MOSI phase.

uint8_t tx_data[4]
    If SPI_TRANS_USE_TXDATA is set, data set here is sent directly from this variable.

void *rx_buffer
    Pointer to receive buffer, or NULL for no MISO phase. Written by 4 bytes-unit if DMA is used.

uint8_t rx_data[4]
    If SPI_TRANS_USE_RXDATA is set, data is received directly to this variable.

struct spi_transaction_ext_t
    This struct is for SPI transactions which may change their address and command length. Please do set the flags in base to SPI_TRANS_VARIABLE_CMD_ADR to use the bit length here.

Public Members
struct spi_transaction_t

Transaction data, so that pointer to spi_transaction_t can be converted into spi_transaction_ext_t.

uint8_t command_bits

The command length in this transaction, in bits.

uint8_t address_bits

The address length in this transaction, in bits.

uint8_t dummy_bits

The dummy length in this transaction, in bits.

Macros

SPI_MASTER_FREQ_8M

SPI master clock is divided by 80MHz apb clock. Below defines are example frequencies, and are accurate. Be free to specify a random frequency, it will be rounded to closest frequency (to macros below if above 8MHz).

8MHz

SPI_MASTER_FREQ_9M

8.89MHz

SPI_MASTER_FREQ_10M

10MHz

SPI_MASTER_FREQ_11M

11.43MHz

SPI_MASTER_FREQ_13M

13.33MHz

SPI_MASTER_FREQ_16M

16MHz

SPI_MASTER_FREQ_20M

20MHz

SPI_MASTER_FREQ_26M

26.67MHz

SPI_MASTER_FREQ_40M

40MHz

SPI_MASTER_FREQ_80M

80MHz

SPI_DEVICE_TXBIT_LSBFIRST

Transmit command/address/data LSB first instead of the default MSB first.
**Chapter 2. API 参考**

**SPI_DEVICE_RXBIT_LSBFIRST**
Receive data LSB first instead of the default MSB first.

**SPI_DEVICE_BIT_LSBFIRST**
Transmit and receive LSB first.

**SPI_DEVICE_3WIRE**
Use MOSI (=spid) for both sending and receiving data.

**SPI_DEVICE_POSITIVE_CS**
Make CS positive during a transaction instead of negative.

**SPI_DEVICE_HALF Duplex**
Transmit data before receiving it, instead of simultaneously.

**SPI_DEVICE_CLK_AS_CS**
Output clock on CS line if CS is active.

**SPI_DEVICE_NO_DUMMY**
There are timing issue when reading at high frequency (the frequency is related to whether iomux pins are used, valid time after slave sees the clock).

- In half-duplex mode, the driver automatically inserts dummy bits before reading phase to fix the timing issue. Set this flag to disable this feature.
- In full-duplex mode, however, the hardware cannot use dummy bits, so there is no way to prevent data being read from getting corrupted. Set this flag to confirm that you’re going to work with output only, or read without dummy bits at your own risk.

**SPI_DEVICE_DDRCLK**

**SPI_DEVICE_NO_RETURN_RESULT**
Don’t return the descriptor to the host on completion (use post_cb to notify instead)

**SPI_TRANS_MODE_DIO**
Transmit/receive data in 2-bit mode.

**SPI_TRANS_MODE_QIO**
Transmit/receive data in 4-bit mode.

**SPI_TRANS_USE_RXDATA**
Receive into rx_data member of spi_transaction_t instead into memory at rx_buffer.

**SPI_TRANS_USE_TXDATA**
Transmit tx_data member of spi_transaction_t instead of data at tx_buffer. Do not set tx_buffer when using this.

**SPI_TRANS_MODE_DIOQIO_ADDR**
Also transmit address in mode selected by SPI_MODE_DIO/SPI_MODE_QIO.
Chapter 2. API

**SPI_TRANS_VARIABLE_CMD**

Use the `command_bits` in `spi_transaction_ext_t` rather than default value in `spi_device_interface_config_t`.

**SPI_TRANS_VARIABLE_ADDR**

Use the `address_bits` in `spi_transaction_ext_t` rather than default value in `spi_device_interface_config_t`.

**SPI_TRANS_VARIABLE_DUMMY**

Use the `dummy_bits` in `spi_transaction_ext_t` rather than default value in `spi_device_interface_config_t`.

**SPI_TRANS_CS_KEEP_ACTIVE**

Keep CS active after data transfer.

**SPI_TRANS_MULTILINE_CMD**

The data lines used at command phase is the same as data phase (otherwise, only one data line is used at command phase).

**SPI_TRANS_MODE_OCT**

Transmit/receive data in 8-bit mode.

**SPI_TRANS_MULTILINE_ADDR**

The data lines used at address phase is the same as data phase (otherwise, only one data line is used at address phase).

**Type Definitions**

typedef struct `spi_transaction_t` spi_transaction_t

typedef void (*`transaction_cb_t`)(`spi_transaction_t` *trans)

typedef struct `spi_device_t` *`spi_device_handle_t`

Handle for a device on a SPI bus.

2.6.21 SPI 从机驱动程序

SPI 从机驱动程序控制在 ESP32 中作为从机的 SPI 外设。

**ESP32 中 SPI 外设概述**

ESP32 集成了 2 个通用的 SPI 控制器，可用作片外 SPI 主机驱动的从机节点。

- SPI2，有时也称为 HSPI
- SPI3，有时也称为 VSPI

SPI2 和 SPI3 各自具有一个与之同名的独立总线信号。
术语
下表为 SPI 主机驱动的相关术语。

<table>
<thead>
<tr>
<th>术语</th>
<th>定义</th>
</tr>
</thead>
<tbody>
<tr>
<td>主机 (Host)</td>
<td>ESP32 外部的 SPI 控制器外设。用作 SPI 主机，在总线上发起 SPI 传输。</td>
</tr>
<tr>
<td>从机设备 (Device)</td>
<td>SPI 从机设备 (通用 SPI 控制器)。每个从机设备共享 MOSI、MISO 和 SCLK 信号。但是只有当主机向从机设备的专属 CS 线发送信号时，从机设备才在总线上处于激活状态。</td>
</tr>
<tr>
<td>总线 (Bus)</td>
<td>信号总线，由连接到同一主机的所有从机设备共用。一般来说，一条总线包括以下线路：MISO、MOSI、SCLK、一条或多条 CS 线，以及可选的 QUADWP 和 QUADHD。每个从机设备都有单独的 CS 线，除此之外，所有从机设备都连接在同一根总线上。如果以菊花链的方式连接，几个从机设备也可以共享一条 CS 线。</td>
</tr>
<tr>
<td>MISO</td>
<td>主机输入，从机输出。数据从从机设备发送至主机。</td>
</tr>
<tr>
<td>MOSI</td>
<td>主机输出，从机输入。数据从主机发送至从机设备。</td>
</tr>
<tr>
<td>SCLK</td>
<td>串行时钟。由主机产生的振荡信号，使数据位的传输保持同步。</td>
</tr>
<tr>
<td>CS</td>
<td>片选。允许主机选择连接到总线上的单个从机设备，以便发送或接收数据。</td>
</tr>
<tr>
<td>QUADWP</td>
<td>写保护信号。只用于 4 位 (qio/qout) 传输。</td>
</tr>
<tr>
<td>QUADHD</td>
<td>保持信号。只用于 4 位 (qio/qout) 传输。</td>
</tr>
<tr>
<td>断言 (Assertion)</td>
<td>指激活一条线的操作。反之，将线路恢复到非活动状态 (回到空闲状态) 的操作称为去断言。</td>
</tr>
<tr>
<td>传输事务 (Transaction)</td>
<td>即主机断言从机设备的 CS 线，向从机设备传输数据，接着去断言 CS 线的过程。传输事务为原子操作，不可打断。</td>
</tr>
<tr>
<td>发射沿 (Launch Edge)</td>
<td>源寄存器将信号发射到线路上的时钟边沿。</td>
</tr>
<tr>
<td>锁存沿 (Latch Edge)</td>
<td>目的寄存器锁存信号的时钟边沿。</td>
</tr>
</tbody>
</table>

驱动程序的功能
SPI 从机驱动程序允许将 SPI 外设作为全双工设备使用。驱动程序可以发送/接收长度不超过 64 字节的传输事务，或者利用 DMA 来发送/接收更长的传输事务。然而，存在一些与 DMA 有关的已知问题。

SPI 传输事务
主机断言 CS 线并在 SCLK 线上发出时钟脉冲时，一次全双工 SPI 传输事务就此开始。每个时钟脉冲都意味着通过 MOSI 线从主机转移一个数据位到从机设备上，并同时通过 MISO 线返回一个数据位。传输事务结束后，主机去断言 CS 线。

传输事务的属性由作为从机设备的 SPI 外设的配置结构体 spi_slave_interface_config_t 和传输事务配置结构体 spi_slave_transaction_t 决定。

由于并非每次传输事务都需要写入和读取数据，您可以选择配置 spi_transaction_t 为仅 TX，仅 RX 或同时 TX 和 RX 传输事务。如果将 spi_slave_transaction_t::rx_buffer 设置为 NULL，读取阶段将被跳过。如果将 spi_slave_transaction_t::tx_buffer 设置为 NULL，则写入阶段将被跳过。

备注：主机应在从机设备准备好接收数据之后再进行传输事务。建议使用另外一个 GPIO 管脚作为握手信号来同步设备。更多细节，请参阅传输事务间例。
使用驱动程序

- 调用函数 `cpp:func:sipSlaveInitialize`，将 SPI 外设初始化为从机设备。请确保在 `bus_config` 中设置正确的 I/O 管脚，并将未使用的信号设置为 -1。

如果传输事务的数据大于 32 字节，需要将参数 `dma_chan` 分别设置为 1 或 2 以使能 DMA 通道 1 或通道 2。若数据小于 32 字节，则应将 `dma_chan` 设为 0。

- 传输事务开始前，需设置要求的事务参数填充一个或多个 `spi_slave_transaction_t` 结构体。可以调用`spi_slave_queue_trans()` 将所有传输事务推入队列，并在后续使用函数 `spi_slave_get_trans_result()` 查询结果。也可以将所有请求输入 `spi_slave_transmit()` 中单独处理。主机上的传输事务完成前，两个函数将被阻塞，以便发送并接收队列中的数据。

- (可选) 如需卸载 SPI 从机驱动程序，请调用 `spi_slave_free()`。

传输事务数据和主从机长度不匹配

通常，通过从机设备进行传输的数据会被读取或写入到由 `spi_slave_transaction_t::rx_buffer` 和`spi_slave_transaction_t::tx_buffer` 指示的大块内存中。可以配置 SPI 驱动程序，使用 DMA 进行传输。在这种情况下，则必须使用 `pvPortMallocCaps(size, MALLOC_CAP_DMA)` 将缓存区分配到具备 DMA 功能的内存中。

驱动程序可以读取或写入缓存区的数据量取决于 `spi_slave_transaction_t::length`，但其并不会定义一次 SPI 传输的实际长度。传输事务的长度由主机的时钟线和 CS 线决定。且只有在传输事务完成后，才能从 `spi_slave_transaction_t::trans_len` 中读取实际长度。

如果传输长度超过缓存区长度，则只有在 `spi_slave_transaction_t::length` 中指定的初始比特数会被发送和接收。此时，`spi_slave_transaction_t::trans_len` 被设置为实际传输事务长度。若需满足实际传输事务长度的要求，请将 `spi_slave_transaction_t::length` 设置为大于 `spi_slave_transaction_t::trans_len` 预期最大值的值。如果传输长度短于缓存区长度，则只传输与缓存区长度相等的数据。

GPIO 交换矩阵和 IO_MUX

ESP32 的大多数外设信号都直接连接到其专用的 IO_MUX 管脚。不过，也可以使用 GPIO 交换矩阵，将信号路由到任何可用的其他管脚。

如果通过 GPIO 交换矩阵路由了至少一个信号，则所有信号都将通过 GPIO 交换矩阵路由。GPIO 交换矩阵以 80 MHz 的频率对所有信号进行采样，并在 GPIO 和外设之间进行传输。

如果已经配置过驱动程序，所有的 SPI 信号都已路由到专用的 IO_MUX 管脚，或者根本没有连接到任何管脚，那么 GPIO 交换矩阵将被绕过。

GPIO 交换矩阵提高了信号传输的灵活性，但也增大了 MISO 信号的输入延迟，导致违反 MISO 设置时间的可能性更高。如需 SPI 高速运行，请使用专用的 IO_MUX 管脚。

备注：更多有关 MISO 输入延迟对最大时钟频率影响的细节，请参阅 `Timing Considerations`。

下表列出了 SPI 总线的 IO_MUX 管脚。

<table>
<thead>
<tr>
<th>管脚名称</th>
<th>GPIO 编号 (SPI2)</th>
<th>GPIO 编号 (SPI3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS0*</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>SCLK</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>MISO</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>MOSI</td>
<td>13</td>
<td>23</td>
</tr>
<tr>
<td>QUADWP</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>QUADHD</td>
<td>4</td>
<td>21</td>
</tr>
</tbody>
</table>

Submit Document Feedback
速度与时钟

传输事务间隔 ESP32 的 SPI 从机外设是由 CPU 控制的通用从机设备。与专用的从机相比，在内嵌 CPU 的 SPI 从机设备中，预定义寄存器的数量有限，所有的传输事务都必须由 CPU 处理。也就是说，传输和响应并不是实时的，且可能存在明显的延迟。

解决方案为，首先使用函数 `spi_slave_queue_trans()`，然后使用 `spi_slave_get_trans_result()`，来代替 `spi_slave_trans()`，由此一来，可使从设备的响应速度提高一倍。

您也可以配置一个 GPIO 管脚，当从设备开始新一次传输事务前，它将通过该管脚向主机发出信号。示例代码存放在 peripherals/spi_slave 目录下。

时钟频率要求 SPI 从机的工作频率最高可达 10 MHz。如果时钟频率过快或占空比不足 50%，数据就无法被正确识别或接收。

除此之外，在数据方面还有一些额外要求，以满足时间限制：

- 读取 (MOSI): 只有当数据在主机的发射沿准备好时，从机设备才能正确读取数据。此为大多数主机的默认情况。
- 写入 (MISO): MISO 信号的输出延迟应短于半个时钟周期，以确保 MISO 线在下一个锁存沿之前保持稳定。基于时钟同步，不同情况下的输出延迟和频率限制如下。

<table>
<thead>
<tr>
<th></th>
<th>MISO 输出延迟 (ns)</th>
<th>频率限制 (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_MUX</td>
<td>43.75</td>
<td>&lt;11.4</td>
</tr>
<tr>
<td>GPIO 交换阵</td>
<td>68.75</td>
<td>&lt;7.2</td>
</tr>
</tbody>
</table>

注：1. 如果频率达到上限，会导致随机误差。2. 主机和设备之间的时钟不确定性 (12.5 ns) 已被考虑在内。3. 表中的输出延迟为理想情况 (无负载) 下的数据。如果 MISO 管脚负载较重，则输出延迟更长，且最大允许频率更低。

例外情况：如果主机支持更多相关 MISO 设置时的选项，例如，可以在下一个边沿锁存数据，或可以配置锁存时间，则频率限制的上限会更高。

条件和已知问题

1. 若启用了 DMA，则 RX 缓冲区应该以字对齐（从 32 位边界开始，字节长度为 4 的倍数）。否则，DMA 可能无法正确写入或无法实现边界对齐。若此项条件不满足，驱动程序将会报错。

2. 此外，使用 DMA 时需启用 SPI 模式 1 和模式 3。在 SPI 模式 0 和模式 2 下，为满足时序要求，MISO 信号必须提前半个时钟周期启动。新的时序如下：

如果启用了 DMA，从机设备的发送沿会比正常时间提前半个 SPI 时钟周期，变为主机的实际锁存沿。在这种情况下，如果 GPIO 交换矩阵被绕过，数据采样的保持时间将是 68.75 ns，而非半个 SPI 时钟周期。如果使用了 GPIO 交换矩阵，保持时间将增加到 93.75 ns。主机应在锁存沿立即采样数据，或在 SPI 模式 1 或模式 3 中进行通信。如果您的主机无法满足上述时间要求，请在没有 DMA 的情况下初始化从机设备。

应用示例

从机设备/主机通信的示例代码存放在 ESP-IDF 示例项目的 peripherals/spi_slave 目录下。
API 参考

Header File

- components/driver/include/driver/spi_slave.h

Functions

**esp_err_t spi_slave_initialize (spi_host_device_t host, const spi_bus_config_t *bus_config, const spi_slave_interface_config_t *slave_config, spi_dma_chan_t dma_chan)**

Initialize a SPI bus as a slave interface.

警告：SPI0/1 is not supported

警告：If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.

警告：The ISR of SPI is always executed on the core which calls this function. Never starve the ISR on this core or the SPI transactions will not be handled.

参数

- **host** - SPI peripheral to use as a SPI slave interface
- **bus_config** - Pointer to a `spi_bus_config_t` struct specifying how the host should be initialized
- **slave_config** - Pointer to a `spi_slave_interface_config_t` struct specifying the details for the slave interface
- **dma_chan** -- Selecting a DMA channel for an SPI bus allows transactions on the bus with size only limited by the amount of internal memory.
  - Selecting SPI_DMA_DISABLED limits the size of transactions.
  - Set to SPI_DMA_DISABLED if only the SPI flash uses this bus.
  - Set to SPI_DMA_CH_AUTO to let the driver to allocate the DMA channel.

返回

- ESP_ERR_INVALID_ARG if configuration is invalid
- ESP_ERR_INVALID_STATE if host already is in use
- ESP_ERR_NOT_FOUND if there is no available DMA channel
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

**esp_err_t spi_slave_free (spi_host_device_t host)**

Free a SPI bus claimed as a SPI slave interface.

参数 **host** - SPI peripheral to free

返回

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if not all devices on the bus are freed
- ESP_OK on success

**esp_err_t spi_slave_queue_trans (spi_host_device_t host, const spi_slave_transaction_t *trans_desc, TickType_t ticks_to_wait)**

Queue a SPI transaction for execution.

Queue a SPI transaction to be executed by this slave device. (The transaction queue size was specified when the slave device was initialised via `spi_slave_initialize`.) This function may block if the queue is full (depending on the `ticks_to_wait` parameter). No SPI operation is directly initiated by this function, the next queued transaction will happen when the master initiates a SPI transaction by pulling down CS and sending out clock signals.
This function hands over ownership of the buffers in `trans_desc` to the SPI slave driver; the application is not to access this memory until `spi_slave_queue_trans` is called to hand ownership back to the application.

### Parameters
- **host** - SPI peripheral that is acting as a slave
- **trans_desc** - Description of transaction to execute. Not const because we may want to write status back into the transaction description.
- **ticks_to_wait** - Ticks to wait until there’s room in the queue; use port MAX_DELAY to never time out.

### Return
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

**esp_err_t spi_slave_get_trans_result (spi_host_device_t host, spi_slave_transaction_t **trans_desc, TickType_t ticks_to_wait)**

Get the result of a SPI transaction queued earlier.

This routine will wait until a transaction to the given device (queued earlier with `spi_slave_queue_trans`) has successfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or re-use the buffers.

It is mandatory to eventually use this function for any transaction queued by `spi_slave_queue_trans`.

### Parameters
- **host** - SPI peripheral that is acting as a slave
- **trans_desc** [out] Pointer to variable able to contain a pointer to the description of the transaction that is executed
- **ticks_to_wait** - Ticks to wait until there’s a returned item; use port MAX_DELAY to never time out.

### Return
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

**esp_err_t spi_slave_transmit (spi_host_device_t host, spi_slave_transaction_t *trans_desc, TickType_t ticks_to_wait)**

Do a SPI transaction.

Essentially does the same as `spi_slave_queue_trans` followed by `spi_slave_get_trans_result`. Do not use this when there is still a transaction queued that hasn’t been finalized using `spi_slave_get_trans_result`.

### Parameters
- **host** - SPI peripheral that is acting as a slave
- **trans_desc** - Pointer to variable able to contain a pointer to the description of the transaction that is executed. Not const because we may want to write status back into the transaction description.
- **ticks_to_wait** - Ticks to wait until there’s a returned item; use port MAX_DELAY to never time out.

### Return
- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

**Structures**

**struct spi_slave_interface_config_t**

This is a configuration for a SPI host acting as a slave device.

**Public Members**
int spics_io_num
   CS GPIO pin for this device.

uint32_t flags
   Bitwise OR of SPI_SLAVE_* flags.

int queue_size
   Transaction queue size. This sets how many transactions can be ‘in the air’ (queued using spi_slave_queue_trans but not yet finished using spi_slave_get_trans_result) at the same time.

uint8_t mode
   SPI mode, representing a pair of (CPOL, CPHA) configuration:
   • 0: (0, 0)
   • 1: (0, 1)
   • 2: (1, 0)
   • 3: (1, 1)

slave_transaction_cb_t post_setup_cb
   Callback called after the SPI registers are loaded with new data.
   This callback is called within interrupt context should be in IRAM for best performance, see “Transferring Speed” section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

slave_transaction_cb_t post_trans_cb
   Callback called after a transaction is done.
   This callback is called within interrupt context should be in IRAM for best performance, see “Transferring Speed” section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

struct spi_slave_transaction_t
   This structure describes one SPI transaction

**Public Members**

size_t length
   Total data length, in bits.

size_t trans_len
   Transaction data length, in bits.

const void *tx_buffer
   Pointer to transmit buffer, or NULL for no MOSI phase.

void *rx_buffer
   Pointer to receive buffer, or NULL for no MISO phase. When the DMA is enabled, must start at WORD boundary (rx_buffer%4==0), and has length of a multiple of 4 bytes.

void *user
   User-defined variable. Can be used to store eg transaction ID.
Chapter 2. API Reference

**Macros**

**SPI_SLAVE_TXBIT_LSBFIRST**
Transmit command/address/data LSB first instead of the default MSB first.

**SPI_SLAVE_RXBIT_LSBFIRST**
Receive data LSB first instead of the default MSB first.

**SPI_SLAVE_BIT_LSBFIRST**
Transmit and receive LSB first.

**SPI_SLAVE_NO_RETURN_RESULT**
Don’t return the descriptor to the host on completion (use `post_trans_cb` to notify instead)

**Type Definitions**

typedef struct spi_slave_transaction_t spi_slave_transaction_t

typedef void (*slave_transaction_cb_t)(spi_slave_transaction_t *trans)

### 2.6.22 ESP32-WROOM-32SE (Secure Element)

**Overview**

The ESP32-WROOM-32SE has integrated Microchip’s ATECC608A cryptoauth chip in the module. ATECC608A is secure element which would generate and store ECC private key in the hardware. The ECC private key can be used to enhance security to connect to IoT cloud services with use of X.509 based mutual authentication. The application example demonstrates ECDSA sign and verify functions using ECC private key stored in ATECC608A

**Application Example**

Secure Element ECDSA Sign/Verify example: peripherals/secure_element/atecc608_ecdsa.

**How to configure and provision ESP32-WROOM-32SE for TLS**

To configure and provision ATECC608A chip on ESP32-WROOM-32SE please visit esp_cryptoauth_utility

**How to use ATECC608A of ESP32-WROOM-32SE for TLS**

ATECC608A can be used for TLS connections using ESP-TLS. To configure ESP-TLS for using secure element please refer ATECC608A with ESP-TLS in ESP-TLS documentation.

### 2.6.23 触摸传感器
概述
触摸传感器系统由保护覆盖层、触摸电极、绝缘基板和走线组成，保护覆盖层位于最上层，绝缘基板上设有电极及走线。用户触摸覆盖层将产生电容变化，根据电容变化判断此次触摸是否为有效触摸行为。

触摸传感器可以以矩阵或滑条等方式组合使用，从而覆盖更大触感区域及更多触感点。触摸传感由软件或专用硬件计时器发起，由有限状态机 (FSM) 硬件控制。

如需了解触摸传感器设计、操作及其控制寄存器等相关信息，请参考《ESP32 技术参考手册》(PDF) 中“片上传感器与模拟信号处理”章节。

请参考 触摸传感器应用方案简介，查看触摸传感器设计详情和固件开发指南。

如果想评估触摸传感器的多种应用场景，请查看 ESP32 触摸功能开发套件。

功能介绍
下面将 API 分解成几个函数组进行介绍，帮助用户快速了解以下功能：

- 初始化触摸传感器驱动程序
- 配置触摸传感器 GPIO 管脚
- 触摸状态测量
- 调整测量参数（优化测量）
- 滤波采样
- 触摸检测方式
- 设置中断信号检测模式动作
- 中断触发，唤醒睡眠模式

请前往API 参考 章节，查看某一函数的具体描述。应用示例 章节则介绍了此 API 的具体实现。

初始化触摸传感器驱动程序 使用触摸传感器之前，需要先调用 touch_pad_init() 函数初始化触摸传感器驱动程序。此函数设置了API 参考 项下的 Macros 中列出的几项 .__DEFAULT 驱动程序参数，同时删除之前设置过的触摸传感器信息 (如有)，并启用中断。

如果不再需要该驱动程序，可以调用 touch_pad_deinit() 释放已初始化的驱动程序。

配置触摸传感器 GPIO 管脚 可调用 touch_pad_config() 使能某一 GPIO 的触感功能。ESP32 最多可支持 10 个电容式触摸传感器通道。

<table>
<thead>
<tr>
<th>触摸传感器通道</th>
<th>GPIO 管脚</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>GPIO4</td>
</tr>
<tr>
<td>T1</td>
<td>GPIO0</td>
</tr>
<tr>
<td>T2</td>
<td>GPIO2</td>
</tr>
<tr>
<td>T3</td>
<td>MTDO</td>
</tr>
<tr>
<td>T4</td>
<td>MTCK</td>
</tr>
<tr>
<td>T5</td>
<td>MTDI</td>
</tr>
<tr>
<td>T6</td>
<td>MTMS</td>
</tr>
<tr>
<td>T7</td>
<td>GPIO27</td>
</tr>
<tr>
<td>T8</td>
<td>32K_XN</td>
</tr>
<tr>
<td>T9</td>
<td>32K_XP</td>
</tr>
</tbody>
</table>

使用 touch_pad_set_fsm_mode() 选择触摸传感器测量（由 FSM 操作）是由硬件定时器自动启动，还是由软件自动启动。如果选择软件模式，请使用 touch_pad_sw_start() 启动 FSM。

触摸状态测量 借助以下两个函数从传感器读取原始数据和滤波后的数据：

- touch_pad_read_raw_data()
- touch_pad_read_filtered()
这两个函数也可以用于检查触碰和释放触摸传感器时传感器读数变化范围，然后根据这些信息设定触摸传感器的触摸阈值。

备注：使用`touch_pad_read_filtered()`之前，需要先调用`filter_data()`中特定的滤波器函数来初始化并配置该滤波器。

请参考应用示例`peripherals/touch_sensor/touch_sensor_v1/touch_pad_read`，查看如何使用读取触摸传感器数据。

测量方式：触摸传感器会统计固定时间内的充电次数，其计数结果即为原始数据。可由`touch_pad_read_raw_data()`读出。上述固定时间可通过`touch_pad_set_measurement_clock_cycles()`设置。完成一次测量后，触摸传感器会在下次测量开始前保持睡眠状态。两次测量之间的间隔时间可由`touch_pad_set_measurement_interval()`进行设置。

备注：若设置的计数时间太短（即测量持续的时钟周期数太小），则可能导致结果不准确；但是过大的计数时间也会造成功耗上升。另外，若睡眠时间加测量时间的总时间过长，则会造成触摸传感器响应变慢。

优化测量：触摸传感器设有数个可配置参数，以适应触摸传感器设计特点。例如，如果需要感知较细微的电容变化，则可以缩小触摸传感器充电放电的参考电压范围。用户可以使用`touch_pad_set_voltage()`函数设置电压参考低值和参考高值。

优化测量除了可以识别细微的电容变化之外，还可以降低应用功耗，但可能会增加测量噪声干扰。如果得到的动态读数范围结果比较理想，则可以调用`touch_pad_set_measurement_clock_cycles()`函数来减少测量时间，从而进一步降低功耗。

可用的测量参数及相应的‘set’函数总结如下：

- 触摸传感器充电放电参数：
  - 电压门限：`touch_pad_set_voltage()`
  - 速率（斜率）：`touch_pad_set_cnt_mode()`

- 单次测量所用的时钟周期：`touch_pad_set_measurement_clock_cycles()`

电压门限（参考低值/参考高值）、速率（斜率）与测量时间的关系如下图所示：

![图20：触摸传感器 - 测量参数之间的关系](image-url)

Espressif Systems 1266 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
Chapter 2. API 参考

上图中的 Output 代表触碰传感器读值，即一个测量周期内测得的脉冲计数值。
所有函数均成对出现，用于设定某一特定参数，并获取当前参数值。例如：`touch_pad_set_voltage()` 和 `touch_pad_get_voltage()`。

### 波动采样
如果测量中存在噪声，可以使用提供的 API 函数对采样进行滤波。使用滤波器之前，请先调用 `touch_pad_filter_start()` 启动该滤波器。
滤波器类型为 IIR（无限脉冲响应滤波器），用户可以调用 `touch_pad_set_filter_period()` 配置此类滤波器的采样周期。
如需停止滤波器，请调用 `touch_pad_filter_stop()` 函数。如果不再使用该滤波器，请调用 `touch_pad_filter_delete()` 删除此滤波器。

### 触摸监测
触摸监测基于用户配置的阈值和 FSM 执行的原始测量，并由 ESP32 硬件实现。用户可以调用 `touch_pad_get_status()` 查看被触碰的触摸传感器，或调用 `touch_pad_clear_status()` 清除触摸状态信息。
用户也可以将硬件触摸监测连接至中断，详细介绍见下一章节。
如果测量中存在噪声，且电容变化幅度较小，硬件触摸监测结果可能就不太理想。如需解决这一问题，不建议使用硬件监测或中断信号，建议用户在自己的应用程序中进行采样滤波，并执行触摸监测。参考 [peripherals/touch_sensor/touch_sensor_v1/touch_pad_interrupt]。(查看以上两种触摸监测的实现方式)

### 中断触发
在对触摸监测启用中断之前，请先设置一个触摸检测阈值。然后使用 [触摸状态测量] 中所述的函数读取并显示触摸和释放触摸传感器时测得的结果。如果测量中存在噪声且相对电容变化较小，请使用滤波器。用户也可以根据应用程序和环境条件，测试温度和电源电压变化对测量值的影响。
确定监测阈值后就可以在初始化时调用 `touch_pad_config()` 设置此阈值，或在运行时调用 `touch_pad_set_thresh()` 设置此阈值。
下一步就是设置如何触发中断。用户可以设置在阈值以下或以上触发中断，具体触发模式由函数 `touch_pad_set_trigger_mode()` 设置。
最后用户可以使用以下函数配置和管理中断调用：
- `touch_pad_isr_register()` / `touch_pad_isr_deregister()`
- `touch_pad_intr_enable()` / `touch_pad_intr_disable()`
中断配置完成后，用户可以调用 `touch_pad_get_status()`，查看中断信号来自哪个触摸传感器，也可以调用 `touch_pad_clear_status()` 清除触摸传感器状态信息。备注：触摸监测中的中断信号基于原始/未经滤波的采样（对比用户设置的阈值），并在硬件中实现。启用软件滤波 API(请参考 波动采样) 并不会影响这一过程。

### 从睡眠模式唤醒
如果使用触摸传感器中断将芯片从睡眠模式唤醒，用户可以选择配置一些触摸传感器，例如 SET1 或 SET2 和 SET2。触摸这些触摸传感器将触发中断并唤醒芯片，请调用 `touch_pad_set_trigger_source()` 实现上述操作。
用户可以使用以下函数管理 ‘SET’ 中触摸传感器所需的位模式配置：
- `touch_pad_set_group_mask()` / `touch_pad_get_group_mask()`
- `touch_pad_clear_group_mask()`

### 应用示例
- 触摸传感器读值示例： `peripherals/touch_sensor/touch_sensor_v1/touch_pad_read`
- 触摸传感器中断示例： `peripherals/touch_sensor/touch_sensor_v1/touch_pad_interrupt`
Chapter 2. API 参考

API 参考

Header File

- components/driver/esp32/include/driver/touch_sensor.h

Functions

`esp_err_t touch_pad_config (touch_pad_t touch_num, uint16_t threshold)`

Configure touch pad interrupt threshold.

备注： If FSM mode is set to TOUCH_FSM_MODE_TIMER, this function will be blocked for one measurement cycle and wait for data to be valid.

参数

- `touch_num` – touch pad index
- `threshold` – interrupt threshold

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG if argument wrong
- ESP_FAIL if touch pad not initialized

`esp_err_t touch_pad_read (touch_pad_t touch_num, uint16_t *touch_value)`

get touch sensor counter value. Each touch sensor has a counter to count the number of charge/discharge cycles. When the pad is not ‘touched’, we can get a number of the counter. When the pad is ‘touched’, the value in counter will get smaller because of the larger equivalent capacitance.

备注： This API requests hardware measurement once. If IIR filter mode is enabled, please use ‘touch_pad_read_raw_data’ interface instead.

参数

- `touch_num` – touch pad index
- `touch_value` – pointer to accept touch sensor value

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Touch pad parameter error
- ESP_ERR_INVALID_STATE This touch pad hardware connection is error, the value of “touch_value” is 0.
- ESP_FAIL Touch pad not initialized

`esp_err_t touch_pad_read_filtered (touch_pad_t touch_num, uint16_t *touch_value)`

get filtered touch sensor counter value by IIR filter.

备注： touch_pad_filter_start has to be called before calling touch_pad_read_filtered. This function can be called from ISR

参数

- `touch_num` – touch pad index
- `touch_value` – pointer to accept touch sensor value

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG Touch pad parameter error
- ESP_ERR_INVALID_STATE This touch pad hardware connection is error, the value of “touch_value” is 0.
## Chapter 2. API

- ESP_FAIL Touch pad not initialized

```c
esp_err_t touch_pad_read_raw_data(touch_pad_t touch_num, uint16_t *touch_value)
```

Get raw data (touch sensor counter value) from IIR filter process. Need not request hardware measurements.

**备注:** touch_pad_filter_start has to be called before calling touch_pad_read_raw_data. This function can be called from ISR.

### 参数
- `touch_num` – touch pad index
- `touch_value` – pointer to accept touch sensor value

### 返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Touch pad parameter error
- ESP_ERR_INVALID_STATE This touch pad hardware connection is error, the value of “touch_value” is 0.
- ESP_FAIL Touch pad not initialized

```c
esp_err_t touch_pad_set_filter_read_cb(filter_cb_t read_cb)
```

Register the callback function that is called after each IIR filter calculation.

**备注:** The ‘read_cb’ callback is called in timer task in each filtering cycle.

### 参数 `read_cb` – Pointer to filtered callback function. If the argument passed in is NULL, the callback will stop.

### 返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG set error

```c
esp_err_t touch_pad_isr_register(intr_handler_t fn, void *arg)
```

Register touch-pad ISR. The handler will be attached to the same CPU core that this function is running on.

### 参数
- `fn` – Pointer to ISR handler
- `arg` – Parameter for ISR

### 返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO error
- ESP_ERR_NO_MEM No memory

```c
esp_err_t touch_pad_set_measurement_clock_cycles(uint16_t clock_cycle)
```

Set the clock cycles of each measurement.

**备注:** This function will specify the clock cycles of each measurement and the clock is sourced from SOC_MOD_CLK_RTC_FAST, its default frequency is SOC_CLK_RC_FAST_FREQ_APPROX The touch sensor will record the charge and discharge times during these clock cycles as the final result (raw value).

**备注:** If clock cycles is too small, it may lead to inaccurate results.

### 参数 `clock_cycle` – The clock cycles of each measurement measure_time = clock_cycle / SOC_CLK_RC_FAST_FREQ_APPROX, the maximum measure time is 0xffff / SOC_CLK_RC_FAST_FREQ_APPROX
Chapter 2. API

**API Reference**

### ESP_OK

Set the clock cycle success

**esp_err_t touch_pad_get_measurement_clock_cycles (uint16_t *clock_cycle)**

Get the clock cycles of each measurement.

**参数**
- `clock_cycle` - The clock cycles of each measurement

**返回**
- ESP_OK Get the clock cycle success
- ESP_ERR_INVALID_ARG The input parameter is NULL

**esp_err_t touch_pad_set_measurement_interval (uint16_t interval_cycle)**

Set the interval between two measurements.

**备注：** The touch sensor will sleep between two measurements. This function is to set the interval cycle. And the interval is clocked from SOC_MOD_CLK_RTC_SLOW, its default frequency is SOC_CLK_RC_SLOW_FREQ_APPROX.

**参数**
- `interval_cycle` - The interval between two measurements

**返回**
- ESP_OK Set interval cycle success

**esp_err_t touch_pad_get_measurement_interval (uint16_t *interval_cycle)**

Get the interval between two measurements.

**参数**
- `interval_cycle` - The interval between two measurements

**返回**
- ESP_OK Get interval cycle success
- ESP_ERR_INVALID_ARG The input parameter is NULL

**esp_err_t touch_pad_set_meas_time (uint16_t sleep_cycle, uint16_t meas_cycle)**

Set touch sensor measurement and sleep time. Excessive total time will slow down the touch response. Too small measurement time will not be sampled enough, resulting in inaccurate measurements.

**备注：** The touch sensor will count the number of charge/discharge cycles over a fixed period of time (specified as the second parameter). That means the number of cycles (raw value) will decrease as the capacity of the touch pad is increasing.

**备注：** The greater the duty cycle of the measurement time, the more system power is consumed.

**参数**
- `sleep_cycle` - The touch sensor will sleep after each measurement. `sleep_cycle` decide the interval between each measurement. `t_sleep = sleep_cycle / SOC_CLK_RC_SLOW_FREQ_APPROX`. The approximate frequency value of RTC_SLOW_CLK can be obtained using rtc_clk_slow_freq_get_hz function.
- `meas_cycle` - The duration of the touch sensor measurement. `t_meas = meas_cycle / SOC_CLK_RC_FAST_FREQ_APPROX`. The maximum measure time is 0xffff / SOC_CLK_RC_FAST_FREQ_APPROX

**返回**
- ESP_OK on success
Get touch sensor measurement and sleep time.

- `sleep_cycle` -- Pointer to accept sleep cycle number
- `meas_cycle` -- Pointer to accept measurement cycle count.

*ESP_OK* on success
*ESP_ERR_INVALID_ARG* if the input parameter is NULL

Trigger a touch sensor measurement, only support in SW mode of FSM.

*ESP_OK* on success

Set touch sensor interrupt threshold.

- `touch_num` -- Touch pad index
- `threshold` -- Threshold of touchpad count, refer to `touch_pad_set_trigger_mode` to see how to set trigger mode.

*ESP_OK* on success
*ESP_ERR_INVALID_ARG* if argument is wrong

Get touch sensor interrupt threshold.

- `touch_num` -- Touch pad index
- `threshold` -- Pointer to accept threshold

*ESP_OK* on success
*ESP_ERR_INVALID_ARG* if argument is wrong

Set touch sensor interrupt trigger mode. Interrupt can be triggered either when counter result is less than threshold or when counter result is more than threshold.

- `mode` -- Touch sensor interrupt trigger mode

*ESP_OK* on success
*ESP_ERR_INVALID_ARG* if argument is wrong

Get touch sensor interrupt trigger mode.

- `mode` -- Pointer to accept touch sensor interrupt trigger mode

*ESP_OK* on success

Set touch sensor interrupt trigger source. There are two sets of touch signals. Set1 and set2 can be mapped to several touch signals. Either set will be triggered if at least one of its touch signal is 'touched'. The interrupt can be configured to be generated if set1 is triggered, or only if both sets are triggered.

- `src` -- Touch sensor interrupt trigger source

*ESP_OK* on success
*ESP_ERR_INVALID_ARG* if argument is wrong
esp_err_t touch_pad_get_trigger_source (touch_trigger_src_t *src)
Get touch sensor interrupt trigger source.
参数 src – pointer to accept touch sensor interrupt trigger source
返回 • ESP_OK on success

esp_err_t touch_pad_set_group_mask (uint16_t set1_mask, uint16_t set2_mask, uint16_t en_mask)
Set touch sensor group mask. Touch pad module has two sets of signals, ‘Touched’ signal is triggered only if at least one of touch pad in this group is “touched”. This function will set the register bits according to the given bitmask.
参数
• set1_mask – bitmask of touch sensor signal group1, it’s a 10-bit value
• set2_mask – bitmask of touch sensor signal group2, it’s a 10-bit value
• en_mask – bitmask of touch sensor work enable, it’s a 10-bit value
返回 • ESP_OK on success
• ESP_ERR_INVALID_ARG if argument is wrong

esp_err_t touch_pad_get_group_mask (uint16_t *set1_mask, uint16_t *set2_mask, uint16_t *en_mask)
Get touch sensor group mask.
参数
• set1_mask – pointer to accept bitmask of touch sensor signal group1, it’s a 10-bit value
• set2_mask – pointer to accept bitmask of touch sensor signal group2, it’s a 10-bit value
• en_mask – pointer to accept bitmask of touch sensor work enable, it’s a 10-bit value
返回 • ESP_OK on success

esp_err_t touch_pad_clear_group_mask (uint16_t set1_mask, uint16_t set2_mask, uint16_t en_mask)
Clear touch sensor group mask. Touch pad module has two sets of signals, Interrupt is triggered only if at least one of touch pad in this group is “touched”. This function will clear the register bits according to the given bitmask.
参数
• set1_mask – bitmask touch sensor signal group1, it’s a 10-bit value
• set2_mask – bitmask touch sensor signal group2, it’s a 10-bit value
• en_mask – bitmask of touch sensor work enable, it’s a 10-bit value
返回 • ESP_OK on success
• ESP_ERR_INVALID_ARG if argument is wrong

esp_err_t touch_pad_intr_enable (void)
To enable touch pad interrupt.
返回 • ESP_OK on success

esp_err_t touch_pad_intr_disable (void)
To disable touch pad interrupt.
返回 • ESP_OK on success

esp_err_t touch_pad_intr_clear (void)
To clear touch pad interrupt.
返回 • ESP_OK on success

esp_err_t touch_pad_set_filter_period (uint32_t new_period_ms)
set touch pad filter calibration period, in ms. Need to call touch_pad_filter_start before all touch filter APIs
**Chapter 2. API 参考**

**函数** `touch_pad_get_filter_period` (uint32_t *p_period_ms)

get touch pad filter calibration period, in ms Need to call `touch_pad_get_filter_period` before all touch filter APIs

参数 `p_period_ms` - pointer to accept period

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE driver state error
- ESP_ERR_INVALID_ARG parameter error

**函数** `touch_pad_filter_start` (uint32_t filter_period_ms)

start touch pad filter function This API will start a filter to process the noise in order to prevent false triggering when detecting slight change of capacitance. Need to call `touch_pad_filter_start` before all touch filter APIs

参数 `filter_period_ms` - filter calibration period, in ms

返回

- ESP_OK Success
- ESP_ERR_INVALID_ARG parameter error
- ESP_ERR_NO_MEM No memory for driver
- ESP_ERR_INVALID_STATE driver state error

**函数** `touch_pad_filter_stop` (void)

stop touch pad filter function Need to call `touch_pad_filter_start` before all touch filter APIs

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE driver state error

**函数** `touch_pad_filter_delete` (void)

delete touch pad filter driver and release the memory Need to call `touch_pad_filter_start` before all touch filter APIs

返回

- ESP_OK Success
- ESP_ERR_INVALID_STATE driver state error

**Type Definitions**

typedef void (*filter_cb_t)(uint16_t*raw_value, uint16_t*filtered_value)

Callback function that is called after each IIR filter calculation.

**备注:** This callback is called in timer task in each filtering cycle.

**备注:** This callback should not be blocked.
Param **raw_value**  The latest raw data (touch sensor counter value) that points to all channels ([raw_value[0..TOUCH_PAD_MAX-1]]).

Param **filtered_value**  The latest IIR filtered data (calculated from raw data) that points to all channels ([filtered_value[0..TOUCH_PAD_MAX-1]]).

**Header File**

- components/driver/include/driver/touch_sensor_common.h

**Functions**

```c
esp_err_t touch_pad_init (void)
```

Initialize touch module.

**备注:** If default parameter don’t match the usage scenario, it can be changed after this function.

- ESP_OK Success
- ESP_ERR_NO_MEM Touch pad init error
- ESP_ERR_NOT_SUPPORTED Touch pad is providing current to external XTAL

```c
esp_err_t touch_pad_deinit (void)
```

Un-install touch pad driver.

**备注:** After this function is called, other touch functions are prohibited from being called.

- ESP_OK Success
- ESP_FAIL Touch pad driver not initialized

```c
esp_err_t touch_pad_io_init (touch_pad_t touch_num)
```

Initialize touch pad GPIO.

**参数**  `touch_num` – touch pad index

**备注**
- ESP_OK on success
- ESP_ERR_INVALID_ARG if argument is wrong

```c
esp_err_t touch_pad_set_voltage (touch_high_volt_t refh, touch_low_volt_t refl, touch_volt_atten_t atten)
```

Set touch sensor high voltage threshold of change. The touch sensor measures the channel capacitance value by charging and discharging the channel. So the high threshold should be less than the supply voltage.

**参数**
- `refh` – the value of DREFH
- `refl` – the value of DREFL
- `atten` – the attenuation on DREFH

**备注**
- ESP_OK on success
- ESP_ERR_INVALID_ARG if argument is wrong

```c
esp_err_t touch_pad_get_voltage (touch_high_volt_t *refh, touch_low_volt_t *refl, touch_volt_atten_t *atten)
```

Get touch sensor reference voltage.

**参数**
- `refh` – pointer to accept DREFH value
- `refl` – pointer to accept DREFL value
Chapter 2. API

- `atten` – pointer to accept the attenuation on DREFH
  ESP_OK on success

`esp_err_t touch_pad_set_cnt_mode(touch_pad_t touch_num, touch_cnt_slope_t slope, touch_tie_opt_t opt)`

Set touch sensor charge/discharge speed for each pad. If the slope is 0, the counter would always be zero. If the slope is 1, the charging and discharging would be slow, accordingly. If the slope is set 7, which is the maximum value, the charging and discharging would be fast.

备注：The higher the charge and discharge current, the greater the immunity of the touch channel, but it will increase the system power consumption.

参数
- `touch_num` – touch pad index
- `slope` – touch pad charge/discharge speed
- `opt` – the initial voltage

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if argument is wrong

`esp_err_t touch_pad_get_cnt_mode(touch_pad_t touch_num, touch_cnt_slope_t *slope, touch_tie_opt_t *opt)`

Get touch sensor charge/discharge speed for each pad.

参数
- `touch_num` – touch pad index
- `slope` – pointer to accept touch pad charge/discharge slope
- `opt` – pointer to accept the initial voltage

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if argument is wrong

`esp_err_t touch_pad_isr_deregister(void (*fn)(void*), void* arg)`

Deregister the handler previously registered using touch_pad_isr_handler_register.

参数
- `fn` – handler function to call (as passed to touch_pad_isr_handler_register)
- `arg` – argument of the handler (as passed to touch_pad_isr_handler_register)

返回
- ESP_OK on success
- ESP_ERR_INVALID_STATE if a handler matching both fn and arg isn’t registered

`esp_err_t touch_pad_get_wakeup_status(touch_pad_t *pad_num)`

Get the touch pad which caused wakeup from deep sleep.

参数 `pad_num` – pointer to touch pad which caused wakeup

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG parameter is NULL

`esp_err_t touch_pad_set_fsm_mode(touch_fsm_mode_t mode)`

Set touch sensor FSM mode, the test action can be triggered by the timer, as well as by the software.

参数 `mode` – FSM mode

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if argument is wrong
**API Reference**

**Function:**

```c
esp_err_t touch_pad_get_fsm_mode (touch_fsm_mode_t *mode)
```

Get touch sensor FSM mode.

- **Parameters:**
  - `mode`: pointer to accept FSM mode

- **Return:**
  - ESP_OK on success

**Function:**

```c
esp_err_t touch_pad_clear_status (void)
```

To clear the touch sensor channel active status.

---

**备注:** The FSM automatically updates the touch sensor status. It is generally not necessary to call this API to clear the status.

---

**Function:**

```c
uint32_t touch_pad_get_status (void)
```

Get the touch sensor channel active status mask. The bit position represents the channel number. The 0/1 status of the bit represents the trigger status.

- **Return:**
  - The touch sensor status. e.g. Touch1 trigger status is `status_mask & (BIT1)`.

**Function:**

```c
bool touch_pad_meas_is_done (void)
```

Check touch sensor measurement status.

- **Return:**
  - True measurement is under way
  - False measurement done

**GPIO 宏查找表**

用户可以使用宏定义某一触摸传感器通道的 GPIO，或定义某一 GPIO 的通道。例如:

1. `TOUCH_PAD_NUM5_GPIO_NUM` 定义了通道 5 的 GPIO（即 GPIO 12）;
2. `TOUCH_PAD_GPIO4_CHANNEL` 定义了 GPIO 4 的通道（即通道 0）。

**Header File**

- `components/soc/esp32/include/soc/touch_sensor_channel.h`

**Macros**

- `TOUCH_PAD_GPIO4_CHANNEL`
- `TOUCH_PAD_NUM0_GPIO_NUM`
- `TOUCH_PAD_GPIO0_CHANNEL`
- `TOUCH_PAD_NUM1_GPIO_NUM`
- `TOUCH_PAD_GPIO2_CHANNEL`
- `TOUCH_PAD_NUM2_GPIO_NUM`
- `TOUCH_PAD_GPIO15_CHANNEL`
Chapter 2. API

```
TOUCH_PAD_NUM3_GPIO_NUM
TOUCH_PAD_GPIO13_CHANNEL
TOUCH_PAD_NUM4_GPIO_NUM
TOUCH_PAD_GPIO12_CHANNEL
TOUCH_PAD_NUM5_GPIO_NUM
TOUCH_PAD_GPIO14_CHANNEL
TOUCH_PAD_NUM6_GPIO_NUM
TOUCH_PAD_GPIO27_CHANNEL
TOUCH_PAD_NUM7_GPIO_NUM
TOUCH_PAD_GPIO33_CHANNEL
TOUCH_PAD_NUM8_GPIO_NUM
TOUCH_PAD_GPIO32_CHANNEL
TOUCH_PAD_NUM9_GPIO_NUM

Header File

  • components/hal/include/hal/touch_sensor_types.h

Macros

TOUCH_PAD_BIT_MASK_ALL
TOUCH_PAD_SLOPE_DEFAULT
TOUCH_PAD_TIE_OPT_DEFAULT
TOUCH_PAD_BIT_MASK_MAX
TOUCH_PAD_HIGH_VOLTAGE_THRESHOLD
TOUCH_PAD_LOW_VOLTAGE_THRESHOLD
TOUCH_PAD_ATTEN_VOLTAGE_THRESHOLD
```
Chapter 2. API

### TOUCH_PAD_IDLE_CH_CONNECT_DEFAULT

### TOUCH_PAD_THRESHOLD_MAX
If set touch threshold max value, The touch sensor can’t be in touched status

### TOUCH_PAD_SLEEP_CYCLE_DEFAULT
The timer frequency is RTC_SLOW_CLK (can be 150k or 32k depending on the options), max value is 0xffff

### TOUCH_PAD_MEASURE_CYCLE_DEFAULT
The timer frequency is 8Mhz, the max value is 0x7fff

### TOUCH_FSM_MODE_DEFAULT
The touch FSM may be started by the software or timer

### TOUCH_TRIGGER_MODE_DEFAULT
Interrupts can be triggered if sensor value gets below or above threshold

### TOUCH_TRIGGER_SOURCE_DEFAULT
The wakeup trigger source can be SET1 or both SET1 and SET2

#### Enumerations

```c
enum touch_pad_t
{
    Touch pad channel

    Values:

    enumerator TOUCH_PAD_NUM0
        Touch pad channel 0 is GPIO4(ESP32)

    enumerator TOUCH_PAD_NUM1
        Touch pad channel 1 is GPIO0(ESP32) / GPIO1(ESP32-S2)

    enumerator TOUCH_PAD_NUM2
        Touch pad channel 2 is GPIO2(ESP32) / GPIO2(ESP32-S2)

    enumerator TOUCH_PAD_NUM3
        Touch pad channel 3 is GPIO15(ESP32) / GPIO3(ESP32-S2)

    enumerator TOUCH_PAD_NUM4
        Touch pad channel 4 is GPIO13(ESP32) / GPIO4(ESP32-S2)

    enumerator TOUCH_PAD_NUM5
        Touch pad channel 5 is GPIO12(ESP32) / GPIO5(ESP32-S2)

    enumerator TOUCH_PAD_NUM6
        Touch pad channel 6 is GPIO14(ESP32) / GPIO6(ESP32-S2)
}
```
enumerator TOUCH_PAD_NUM7

Touch pad channel 7 is GPIO27(ESP32) / GPIO7(ESP32-S2)

enumerator TOUCH_PAD_NUM8

Touch pad channel 8 is GPIO33(ESP32) / GPIO8(ESP32-S2)

enumerator TOUCH_PAD_NUM9

Touch pad channel 9 is GPIO32(ESP32) / GPIO9(ESP32-S2)

enumerator TOUCH_PAD_MAX

def touch_high_volt_t

Touch sensor high reference voltage

Values:

enumerator TOUCH_HVOLT_KEEP

Touch sensor high reference voltage, no change

enumerator TOUCH_HVOLT_2V4

Touch sensor high reference voltage, 2.4V

enumerator TOUCH_HVOLT_2V5

Touch sensor high reference voltage, 2.5V

enumerator TOUCH_HVOLT_2V6

Touch sensor high reference voltage, 2.6V

enumerator TOUCH_HVOLT_2V7

Touch sensor high reference voltage, 2.7V

enumerator TOUCH_HVOLT_MAX

def touch_low_volt_t

Touch sensor low reference voltage

Values:

enumerator TOUCH_LVOLT_KEEP

Touch sensor low reference voltage, no change

enumerator TOUCH_LVOLT_0V5

Touch sensor low reference voltage, 0.5V

enumerator TOUCH_LVOLT_0V6

Touch sensor low reference voltage, 0.6V

enumerator TOUCH_LVOLT_0V7

Touch sensor low reference voltage, 0.7V
enumerator TOUCH_LVOLT_0V8
Touch sensor low reference voltage, 0.8V

enumerator TOUCH_LVOLT_MAX

enum touch_volt_atten_t
Touch sensor high reference voltage attenuation

Values:

enumerator TOUCH_HVOLT_ATTEN_KEEP
Touch sensor high reference voltage attenuation, no change

enumerator TOUCH_HVOLT_ATTEN_1V5
Touch sensor high reference voltage attenuation, 1.5V attenuation

enumerator TOUCH_HVOLT_ATTEN_1V
Touch sensor high reference voltage attenuation, 1.0V attenuation

enumerator TOUCH_HVOLT_ATTEN_0V5
Touch sensor high reference voltage attenuation, 0.5V attenuation

enumerator TOUCH_HVOLT_ATTEN_0V
Touch sensor high reference voltage attenuation, 0V attenuation

enumerator TOUCH_HVOLT_ATTEN_MAX

enum touch_cnt_slope_t
Touch sensor charge/discharge speed

Values:

enumerator TOUCH_PAD_SLOPE_0
Touch sensor charge / discharge speed, always zero

enumerator TOUCH_PAD_SLOPE_1
Touch sensor charge / discharge speed, slowest

enumerator TOUCH_PAD_SLOPE_2
Touch sensor charge / discharge speed

enumerator TOUCH_PAD_SLOPE_3
Touch sensor charge / discharge speed

enumerator TOUCH_PAD_SLOPE_4
Touch sensor charge / discharge speed

enumerator TOUCH_PAD_SLOPE_5
Touch sensor charge / discharge speed
enumerator TOUCH_PAD_SLOPE_6
    Touch sensor charge / discharge speed

enumerator TOUCH_PAD_SLOPE_7
    Touch sensor charge / discharge speed, fast

enumerator TOUCH_PAD_SLOPE_MAX

enum touch_tie_opt_t
    Touch sensor initial charge level
    Values:

    enumerator TOUCH_PAD_TIE_OPT_LOW
        Initial level of charging voltage, low level

    enumerator TOUCH_PAD_TIE_OPT_HIGH
        Initial level of charging voltage, high level

    enumerator TOUCH_PAD_TIE_OPT_MAX

enum touch_fsm_mode_t
    Touch sensor FSM mode
    Values:

    enumerator TOUCH_FSM_MODE_TIMER
        To start touch FSM by timer

    enumerator TOUCH_FSM_MODE_SW
        To start touch FSM by software trigger

    enumerator TOUCH_FSM_MODE_MAX

enum touch_trigger_mode_t
    Values:

    enumerator TOUCH_TRIGGER_BELOW
        Touch interrupt will happen if counter value is less than threshold.

    enumerator TOUCH_TRIGGER_ABOVE
        Touch interrupt will happen if counter value is larger than threshold.

    enumerator TOUCH_TRIGGER_MAX

enum touch_trigger_src_t
    Values:

    enumerator TOUCH_TRIGGER_SOURCE_BOTH
        wakeup interrupt is generated if both SET1 and SET2 are “touched”
enumerator TOUCH_TRIGGER_SOURCE_SET1
wakeup interrupt is generated if SET1 is “touched”

enumerator TOUCH_TRIGGER_SOURCE_MAX

### 2.6.24 Two-Wire Automotive Interface (TWAI)

**Overview**

The Two-Wire Automotive Interface (TWAI) is a real-time serial communication protocol suited for automotive and industrial applications. It is compatible with ISO11898-1 Classical frames, thus can support Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID). The ESP32 contains 1 TWAI controller(s) that can be configured to communicate on a TWAI bus via an external transceiver.

| Warning: The TWAI controller is not compatible with ISO11898-1 FD Format frames, and will interpret such frames as errors. |

This programming guide is split into the following sections:

**Sections**

- [Two-Wire Automotive Interface (TWAI)]
  - Overview
  - TWAI Protocol Summary
  - Signals Lines and Transceiver
  - Driver Configuration
  - Driver Operation
  - Examples
  - API Reference

#### TWAI Protocol Summary

The TWAI is a multi-master, multi-cast, asynchronous, serial communication protocol. TWAI also supports error detection and signalling, and inbuilt message prioritization.

**Multi-master:** Any node on the bus can initiate the transfer of a message.

**Multi-cast:** When a node transmits a message, all nodes on the bus will receive the message (i.e., broadcast) thus ensuring data consistency across all nodes. However, some nodes can selectively choose which messages to accept via the use of acceptance filtering (multi-cast).

**Asynchronous:** The bus does not contain a clock signal. All nodes on the bus operate at the same bit rate and synchronize using the edges of the bits transmitted on the bus.

**Error Detection and Signalling:** Every node will constantly monitor the bus. When any node detects an error, it will signal the detection by transmitting an error frame. Other nodes will receive the error frame and transmit their own error frames in response. This will result in an error detection being propagated to all nodes on the bus.

**Message Priorities:** Messages contain an ID field. If two or more nodes attempt to transmit simultaneously, the node transmitting the message with the lower ID value will win arbitration of the bus. All other nodes will become receivers ensuring that there is at most one transmitter at any time.
**TWAI Messages**  
TWAI Messages are split into Data Frames and Remote Frames. Data Frames are used to deliver a data payload to other nodes, whereas a Remote Frame is used to request a Data Frame from other nodes (other nodes can optionally respond with a Data Frame). Data and Remote Frames have two frame formats known as Extended Frame and Standard Frame which contain a 29-bit ID and an 11-bit ID respectively. A TWAI message consists of the following fields:

- 29-bit or 11-bit ID: Determines the priority of the message (lower value has higher priority).
- Data Length Code (DLC) between 0 to 8: Indicates the size (in bytes) of the data payload for a Data Frame, or the amount of data to request for a Remote Frame.
- Up to 8 bytes of data for a Data Frame (should match DLC).

**Error States and Counters**  
The TWAI protocol implements a feature known as “fault confinement” where a persistently erroneous node will eventually eliminate itself from the bus. This is implemented by requiring every node to maintain two internal error counters known as the Transmit Error Counter (TEC) and the Receive Error Counter (REC). The two error counters are incremented and decremented according to a set of rules (where the counters increase on an error, and decrease on a successful message transmission/reception). The values of the counters are used to determine a node’s error state, namely Error Active, Error Passive, and Bus-Off.

**Error Active:** A node is Error Active when both TEC and REC are less than 128 and indicates that the node is operating normally. Error Active nodes are allowed to participate in bus communications, and will actively signal the detection of any errors by automatically transmitting an Active Error Flag over the bus.

**Error Passive:** A node is Error Passive when either the TEC or REC becomes greater than or equal to 128. Error Passive nodes are still able to take part in bus communications, but will instead transmit a Passive Error Flag upon detection of an error.

**Bus-Off:** A node becomes Bus-Off when the TEC becomes greater than or equal to 256. A Bus-Off node is unable to influence the bus in any manner (essentially disconnected from the bus) thus eliminating itself from the bus. A node will remain in the Bus-Off state until it undergoes bus-off recovery.

**Signals Lines and Transceiver**

The TWAI controller does not contain an integrated transceiver. Therefore, to connect the TWAI controller to a TWAI bus, an external transceiver is required. The type of external transceiver used should depend on the application’s physical layer specification (e.g. using SN65HVD23x transceivers for ISO 11898-2 compatibility).

The TWAI controller’s interface consists of 4 signal lines known as TX, RX, BUS-OFF, and CLKOUT. These four signal lines can be routed through the GPIO Matrix to the ESP32’s GPIO pads.

**TX and RX:** The TX and RX signal lines are required to interface with an external transceiver. Both signal lines represent/interpret a dominant bit as a low logic level (0V), and a recessive bit as a high logic level (3.3V).

**BUS-OFF:** The BUS-OFF signal line is optional and is set to a low logic level (0V) whenever the TWAI controller reaches a bus-off state. The BUS-OFF signal line is set to a high logic level (3.3V) otherwise.

**CLKOUT:** The CLKOUT signal line is optional and outputs a prescaled version of the controller’s source clock.

---

**Driver Configuration**

This section covers how to configure the TWAI driver.
Chapter 2. API 参考

## Operating Modes

The TWAI driver supports the following modes of operations:

**Normal Mode:** The normal operating mode allows the TWAI controller to take part in bus activities such as transmitting and receiving messages/error frames. Acknowledgement from another node is required when transmitting a message.

**No Ack Mode:** The No Acknowledgement mode is similar to normal mode, however acknowledgements are not required for a message transmission to be considered successful. This mode is useful when self testing the TWAI controller (loopback of transmissions).

**Listen Only Mode:** This mode will prevent the TWAI controller from influencing the bus. Therefore, transmission of messages/acknowledgement/error frames will be disabled. However the TWAI controller will still be able to receive messages but will not acknowledge the message. This mode is suited for bus monitor applications.

**Alerts**

The TWAI driver contains an alert feature that is used to notify the application layer of certain TWAI controller or TWAI bus events. Alerts are selectively enabled when the TWAI driver is installed, but can be reconfigured during runtime by calling `twai_reconfigure_alerts()`. The application can then wait for any enabled alerts to occur by calling `twai_read_alerts()`. The TWAI driver supports the following alerts:

![Signal lines of the TWAI controller](image)
### TWAIDriverAlerts

<table>
<thead>
<tr>
<th>Alert Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWAI_ALERT_TX_IDLE</td>
<td>No more messages queued for transmission</td>
</tr>
<tr>
<td>TWAI_ALERT_TX_SUCCESS</td>
<td>The previous transmission was successful</td>
</tr>
<tr>
<td>TWAI_ALERT_RX_DATA</td>
<td>A frame has been received and added to the RX queue</td>
</tr>
<tr>
<td>TWAI_ALERT_BELOW_ERR_WARN</td>
<td>Both error counters have dropped below error warning limit</td>
</tr>
<tr>
<td>TWAI_ALERT_ERR_ACTIVE</td>
<td>TWAI controller has become error active</td>
</tr>
<tr>
<td>TWAI_ALERT_RECOVERY_IN_PROGRESS</td>
<td>TWAI controller is undergoing bus recovery</td>
</tr>
<tr>
<td>TWAI_ALERT_BUS_RECOVERED</td>
<td>TWAI controller has successfully completed bus recovery</td>
</tr>
<tr>
<td>TWAI_ALERT_ARB_LOST</td>
<td>The previous transmission lost arbitration</td>
</tr>
<tr>
<td>TWAI_ALERT_ABOVE_ERR_WARN</td>
<td>One of the error counters has exceeded the error warning limit</td>
</tr>
<tr>
<td>TWAI_ALERT_TX_FAILED</td>
<td>A (Bit, Stuff, CRC, Form, ACK) error has occurred on the bus</td>
</tr>
<tr>
<td>TWAI_ALERT_BELOW_ERR_WARN</td>
<td>The previous transmission has failed</td>
</tr>
<tr>
<td>TWAI_ALERT_RX_QUEUE_FULL</td>
<td>The RX queue is full causing a received frame to be lost</td>
</tr>
<tr>
<td>TWAI_ALERT_ERR_PASS</td>
<td>TWAI controller has become error passive</td>
</tr>
<tr>
<td>TWAI_ALERT_BUS_OFF</td>
<td>Bus-off condition occurred. TWAI controller can no longer influence bus</td>
</tr>
</tbody>
</table>

**备注**: The TWAI controller’s error warning limit is used to preemptively warn the application of bus errors before the error passive state is reached. By default, the TWAI driver sets the error warning limit to **96**. The TWAI_ALERT_ABOVE_ERR_WARN is raised when the TEC or REC becomes larger then or equal to the error warning limit. The TWAI_ALERT_BELOW_ERR_WARN is raised when both TEC and REC return back to values below **96**.

**备注**: When enabling alerts, the **TWAI_ALERT_AND_LOG** flag can be used to cause the TWAI driver to log any raised alerts to UART. However, alert logging is disabled and **TWAI_ALERT_AND_LOG** if the **CONFIG_TWAI_ISR_IN_IRAM** option is enabled (see **Placing ISR into IRAM**).

**备注**: The **TWAI_ALERT_ALL** and **TWAI_ALERT_NONE** macros can also be used to enable/disable all alerts during configuration/reconfiguration.

### Bit Timing

The operating bit rate of the TWAI driver is configured using the `twai_timing_config_t` structure. The period of each bit is made up of multiple time quanta, and the period of a time quantum is determined by a prescaled version of the TWAI controller’s source clock. A single bit contains the following segments in the following order:

1. The **Synchronization Segment** consists of a single time quantum
2. **Timing Segment 1** consists of 1 to 16 time quanta before sample point
3. **Timing Segment 2** consists of 1 to 8 time quanta after sample point

The **Baudrate Prescaler** is used to determine the period of each time quantum by dividing the TWAI controller’s source clock. On the ESP32, the `brp` can be any even number from **2** to **128**. Alternatively, you can decide the resolution of each quantum, by setting `twai_timing_config_t::quanta_resolution_hz` to a non-zero value. In this way, the driver can calculate the underlying `brp` value for you. It’s useful when you set different clock sources but want the bitrate to keep the same.

Supported clock source for a TWAI controller is listed in the `twai_clock_source_t` and can be specified in `twai_timing_config_t::clk_src`.

If the ESP32 is a revision 2 or later chip, the **brp** will also support any multiple of **4** from **132** to **256**, and can be enabled by setting the **CONFIG_ESP32_REV_MIN** to revision 2 or higher.
Chapter 2. API

Bittimingconfigurationfor500kbit/sgivenBRP=8,clocksourcefrequencyis80MHz

The sample point of a bit is located on the intersection of Timing Segment 1 and 2. Enabling Triple Sampling will cause 3 time quanta to be sampled per bit instead of 1 (extra samples are located at the tail end of Timing Segment 1).

The Synchronization Jump Width is used to determine the maximum number of time quanta a single bit time can be lengthened/shortened for synchronization purposes. sjw can range from 1 to 4.

Multiple combinations of brp, tseg_1, tseg_2, and sjw can achieve the same bit rate. Users should tune these values to the physical characteristics of their bus by taking into account factors such as propagation delay, node information processing time, and phase errors.

Bit timing macro initializers are also available for commonly used bit rates. The following macro initializers are provided by the TWAI driver.

- TWAI_TIMING_CONFIG_1MBITS
- TWAI_TIMING_CONFIG_800KBITS
- TWAI_TIMING_CONFIG_500KBITS
- TWAI_TIMING_CONFIG_250KBITS
- TWAI_TIMING_CONFIG_125KBITS
- TWAI_TIMING_CONFIG_100KBITS
- TWAI_TIMING_CONFIG_50KBITS
- TWAI_TIMING_CONFIG_25KBITS

Revision 2 or later of the ESP32 also supports the following bit rates:

- TWAI_TIMING_CONFIG_20KBITS
- TWAI_TIMING_CONFIG_16KBITS
- TWAI_TIMING_CONFIG_12_5KBITS

Acceptance Filter The TWAI controller contains a hardware acceptance filter which can be used to filter messages of a particular ID. A node that filters out a message will not receive the message, but will still acknowledge it. Acceptance filters can make a node more efficient by filtering out messages sent over the bus that are irrelevant to the node. The acceptance filter is configured using two 32-bit values within twai_filter_config_t known as the acceptance code and the acceptance mask.

The acceptance code specifies the bit sequence which a message’s ID, RTR, and data bytes must match in order for the message to be received by the TWAI controller. The acceptance mask is a bit sequence specifying which bits of the acceptance code can be ignored. This allows for messages of different IDs to be accepted by a single acceptance code.

The acceptance filter can be used under Single or Dual Filter Mode. Single Filter Mode will use the acceptance code and mask to define a single filter. This allows for the first two data bytes of a standard frame to be filtered, or the entirety of an extended frame’s 29-bit ID. The following diagram illustrates how the 32-bit acceptance code and mask will be interpreted under Single Filter Mode (Note: The yellow and blue fields represent standard and extended frame formats respectively).

Dual Filter Mode will use the acceptance code and mask to define two separate filters allowing for increased flexibility of ID’s to accept, but does not allow for all 29-bits of an extended ID to be filtered. The following diagram
Disabling TX Queue  The TX queue can be disabled during configuration by setting the `tx_queue_len` member of `twai_general_config_t` to 0. This will allow applications that do not require message transmission to save a small amount of memory when using the TWAI driver.

Placing ISR into IRAM  The TWAI driver’s ISR (Interrupt Service Routine) can be placed into IRAM so that the ISR can still run whilst the cache is disabled. Placing the ISR into IRAM may be necessary to maintain the TWAI driver’s functionality during lengthy cache disabling operations (such as SPI Flash writes, OTA updates etc). Whilst the cache is disabled, the ISR will continue to:

- Read received messages from the RX buffer and place them into the driver’s RX queue.
- Load messages pending transmission from the driver’s TX queue and write them into the TX buffer.

To place the TWAI driver’s ISR, users must do the following:

- Enable the `CONFIG_TWAI_ISR_IN_IRAM` option using `idf.py menuconfig`.
- When calling `twai_driver_install()`, the `intr_flags` member of `twai_general_config_t` should set the `ESP_INTR_FLAG_IRAM` set.

备注:  When the `CONFIG_TWAI_ISR_IN_IRAM` option is enabled, the TWAI driver will no longer log any alerts (i.e., the `TWAI_ALERT_AND_LOG` flag will not have any effect).

ESP32 Errata Workarounds  The ESP32’s TWAI controller contains multiple hardware errata (more details about the errata can be found in the ESP32’s ECO document). Some of these errata are critical, and under specific circumstances, can place the TWAI controller into an unrecoverable state (i.e., the controller gets stuck until it is reset by the CPU).

The TWAI driver contains software workarounds for these critical errata. With these workarounds, the ESP32 TWAI driver can operate normally, albeit with degraded performance. The degraded performance will affect users in the following ways depending on what particular errata conditions are encountered:

- The TWAI driver can occasionally drop some received messages.
- The TWAI driver can be unresponsive for a short period of time (i.e., will not transmit or ACK for 11 bit times or longer).
• If `CONFIG_TWAI_ISR_IN_IRAM` is enabled, the workarounds will increase IRAM usage by approximately 1KB.

The software workarounds are enabled by default and it is recommended that users keep this workarounds enabled.

**Driver Operation**

The TWAI driver is designed with distinct states and strict rules regarding the functions or conditions that trigger a state transition. The following diagram illustrates the various states and their transitions.

![State transition diagram of the TWAI driver](image)

**Driver States**

**Uninstalled**: In the uninstalled state, no memory is allocated for the driver and the TWAI controller is powered OFF.

**Stopped**: In this state, the TWAI controller is powered ON and the TWAI driver has been installed. However the TWAI controller will be unable to take part in any bus activities such as transmitting, receiving, or acknowledging messages.

**Running**: In the running state, the TWAI controller is able to take part in bus activities. Therefore messages can be transmitted/received/acknowledged. Furthermore the TWAI controller will be able to transmit error frames upon detection of errors on the bus.

**Bus-Off**: The bus-off state is automatically entered when the TWAI controller’s Transmit Error Counter becomes greater than or equal to 256. The bus-off state indicates the occurrence of severe errors on the bus or in the TWAI controller. Whilst in the bus-off state, the TWAI controller will be unable to take part in any bus activities. To exit the bus-off state, the TWAI controller must undergo the bus recovery process.

**Recovering**: The recovering state is entered when the TWAI controller undergoes bus recovery. The TWAI controller/TWAI driver will remain in the recovering state until the 128 occurrences of 11 consecutive recessive bits is observed on the bus.
**Message Fields and Flags**  The TWAI driver distinguishes different types of messages by using the various bit field members of the `twai_message_t` structure. These bit field members determine whether a message is in standard or extended format, a remote frame, and the type of transmission to use when transmitting such a message.

These bit field members can also be toggled using the `flags` member of `twai_message_t` and the following message flags:

<table>
<thead>
<tr>
<th>Message Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWAI_MSG_FLAG_EXTD</td>
<td>Message is in Extended Frame Format (29bit ID)</td>
</tr>
<tr>
<td>TWAI_MSG_FLAG_RTR</td>
<td>Message is a Remote Frame (Remote Transmission Request)</td>
</tr>
<tr>
<td>TWAI_MSG_FLAG_SS</td>
<td>Transmit message using Single Shot Transmission (Message will not be re-transmitted upon error or loss of arbitration). Unused for received message.</td>
</tr>
<tr>
<td>TWAI_MSG_FLAG_SELF</td>
<td>Transmit message using Self Reception Request (Transmitted message will also be received by the same node). Unused for received message.</td>
</tr>
<tr>
<td>TWAI_MSG_FLAG_DLC_NON_COMP</td>
<td>Message’s Data length code is larger than 8. This will break compliance with TWAI</td>
</tr>
<tr>
<td>TWAI_MSG_FLAG_NONE</td>
<td>Clears all bit fields. Equivalent to a Standard Frame Format (11bit ID) Data Frame.</td>
</tr>
</tbody>
</table>

**Examples**

**Configuration & Installation**  The following code snippet demonstrates how to configure, install, and start the TWAI driver via the use of the various configuration structures, macro initializers, the `twai_driver_install()` function, and the `twai_start()` function.

```c
#include "driver/gpio.h"
#include "driver/twai.h"

void app_main()
{
    //Initialize configuration structures using macro initializers
    twai_general_config_t g_config = TWAI_GENERAL_CONFIG_DEFAULT(GPIO_NUM_21, GPIO_NUM_22, TWAI_MODE_NORMAL);
    twai_timing_config_t t_config = TWAI_TIMING_CONFIG_500KBITS();
    twai_filter_config_t f_config = TWAI_FILTER_CONFIG_ACCEPT_ALL();

    //Install TWAI driver
    if (twai_driver_install(&g_config, &t_config, &f_config) == ESP_OK) {
        printf("Driver installed\n");
    } else {
        printf("Failed to install driver\n");
        return;
    }

    //Start TWAI driver
    if (twai_start() == ESP_OK) {
        printf("Driver started\n");
    } else {
        printf("Failed to start driver\n");
        return;
    }

    ...
}
```

The usage of macro initializers is not mandatory and each of the configuration structures can be manually.
**Message Transmission**  The following code snippet demonstrates how to transmit a message via the usage of the `twai_message_t` type and `twai_transmit()` function.

```c
#include "driver/twai.h"
...
//Configure message to transmit
twai_message_t message;
message.identifier = 0xAAAA;
message.extd = 1;
message.data_length_code = 4;
for (int i = 0; i < 4; i++) {
    message.data[i] = 0;
}
//Queue message for transmission
if (twai_transmit(&message, pdMS_TO_TICKS(1000)) == ESP_OK) {
    printf("Message queued for transmission\n");
} else {
    printf("Failed to queue message for transmission\n");
}
```

**Message Reception**  The following code snippet demonstrates how to receive a message via the usage of the `twai_message_t` type and `twai_receive()` function.

```c
#include "driver/twai.h"
...
//Wait for message to be received
twai_message_t message;
if (twai_receive(&message, pdMS_TO_TICKS(10000)) == ESP_OK) {
    printf("Message received\n");
} else {
    printf("Failed to receive message\n");
    return;
}
//Process received message
if (message.extd) {
    printf("Message is in Extended Format\n");
} else {
    printf("Message is in Standard Format\n");
}
printf("ID is %d\n", message.identifier);
if (!message.rtr) {
    for (int i = 0; i < message.data_length_code; i++) {
        printf("Data byte %d = %d\n", i, message.data[i]);
    }
}
```

**Reconfiguring and Reading Alerts**  The following code snippet demonstrates how to reconfigure and read TWAI driver alerts via the use of the `twai_reconfigure_alerts()` and `twai_read_alerts()` functions.

```c
#include "driver/twai_reconfigure_alerts.h"
...
//Reconfigure alerts to detect Error Passive and Bus-Off error states
```
Chapter 2. API

```c
uint32_t alerts_to_enable = TWAI_ALERT_ERR_PASS | TWAI_ALERT_BUS_OFF;
if (twai_reconfigure_alerts(alerts_to_enable, NULL) == ESP_OK) {
    printf("Alerts reconfigured\n");
} else {
    printf("Failed to reconfigure alerts");
}

//Block indefinitely until an alert occurs
uint32_t alerts_triggered;
twai_read_alerts(&alerts_triggered, portMAX_DELAY);
```

**Stop and Uninstall** The following code demonstrates how to stop and uninstall the TWAI driver via the use of the `twai_stop()` and `twai_driver_uninstall()` functions.

```c
#include "driver/twai.h"
...

//Stop the TWAI driver
if (twai_stop() == ESP_OK) {
    printf("Driver stopped\n");
} else {
    printf("Failed to stop driver\n");
    return;
}

//Uninstall the TWAI driver
if (twai_driver_uninstall() == ESP_OK) {
    printf("Driver uninstalled\n");
} else {
    printf("Failed to uninstall driver\n");
    return;
}
```

**Multiple ID Filter Configuration** The acceptance mask in `twai_filter_config_t` can be configured such that two or more IDs will be accepted for a single filter. For a particular filter to accept multiple IDs, the conflicting bit positions amongst the IDs must be set in the acceptance mask. The acceptance code can be set to any one of the IDs.

The following example shows how to calculate the acceptance mask given multiple IDs:

```
ID1 = 11'b101 1010 0000
ID2 = 11'b101 1010 0001
ID3 = 11'b101 1010 0100
ID4 = 11'b101 1010 1000
//Acceptance Mask
MASK = 11'b000 0000 1101
```

**Application Examples**

**Network Example:** The TWAI Network example demonstrates communication between two ESP32s using the TWAI driver API. One TWAI node acts as a network master that initiates and ceases the transfer of a data from another node acting as a network slave. The example can be found via `peripherals/twai/twai_network`.

**Alert and Recovery Example:** This example demonstrates how to use the TWAI driver’s alert and bus-off recovery API. The example purposely introduces errors on the bus to put the TWAI controller into the Bus-Off state. An alert is used to detect the Bus-Off state and trigger the bus recovery process. The example can be found via `peripherals/twai/twai_alert_and_recovery`. 
Self Test Example: This example uses the No Acknowledge Mode and Self Reception Request to cause the TWAI controller to send and simultaneously receive a series of messages. This example can be used to verify if the connections between the TWAI controller and the external transceiver are working correctly. The example can be found via peripherals/twai/twai_self_test.

API Reference

Header File

- components/hal/include/hal/twai_types.h

Structures

struct twai_message_t

Structure to store a TWAI message.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extd</td>
<td>Extended Frame Format (29bit ID)</td>
</tr>
<tr>
<td>rtr</td>
<td>Message is a Remote Frame</td>
</tr>
<tr>
<td>ss</td>
<td>Transmit as a Single Shot Transmission. Unused for received.</td>
</tr>
<tr>
<td>self</td>
<td>Transmit as a Self Reception Request. Unused for received.</td>
</tr>
<tr>
<td>dlc_non_comp</td>
<td>Message’s Data length code is larger than 8. This will break compliance with ISO 11898-1</td>
</tr>
<tr>
<td>reserved</td>
<td>Reserved bits</td>
</tr>
<tr>
<td>flags</td>
<td>Deprecated: Alternate way to set bits using message flags</td>
</tr>
<tr>
<td>identifier</td>
<td>11 or 29 bit identifier</td>
</tr>
<tr>
<td>data_length_code</td>
<td>Data length code</td>
</tr>
<tr>
<td>data</td>
<td>Data bytes (not relevant in RTR frame)</td>
</tr>
</tbody>
</table>
struct twai_timing_config_t

Structure for bit timing configuration of the TWAI driver.

备注: Macro initializers are available for this structure

**Public Members**

*twai_clock_source_t* `clk_src`

Clock source, set to 0 or TWAI_CLK_SRC_DEFAULT if you want a default clock source

*uint32_t* `quanta_resolution_hz`

The resolution of one timing quanta, in Hz. Note: the value of `brp` will be reflected by this field if it’s non-zero, otherwise, `brp` needs to be set manually.

*uint32_t* `brp`

Baudrate prescale (i.e., clock divider). Any even number from 2 to 128 for ESP32, 2 to 32768 for non-ESP32 chip. Note: For ESP32 ECO 2 or later, multiples of 4 from 132 to 256 are also supported.

*uint8_t* `tseg_1`

Timing segment 1 (Number of time quanta, between 1 to 16)

*uint8_t* `tseg_2`

Timing segment 2 (Number of time quanta, 1 to 8)

*uint8_t* `sjw`

Synchronization Jump Width (Max time quanta jump for synchronize from 1 to 4)

*bool* `triple_sampling`

Enables triple sampling when the TWAI controller samples a bit

struct twai_filter_config_t

Structure for acceptance filter configuration of the TWAI driver (see documentation)

备注: Macro initializers are available for this structure

**Public Members**

*uint32_t* `acceptance_code`

32-bit acceptance code

*uint32_t* `acceptance_mask`

32-bit acceptance mask

*bool* `single_filter`

Use Single Filter Mode (see documentation)
Macros

**TWAI_EXTD_ID_MASK**
- TWAI Constants.
  - Bit mask for 29 bit Extended Frame Format ID

**TWAI_STD_ID_MASK**
- Bit mask for 11 bit Standard Frame Format ID

**TWAI_FRAME_MAX_DLC**
- Max data bytes allowed in TWAI

**TWAI_FRAME_EXTD_ID_LEN_BYTES**
- EFF ID requires 4 bytes (29bit)

**TWAI_FRAME_STD_ID_LEN_BYTES**
- SFF ID requires 2 bytes (11bit)

**TWAI_ERR_PASS_THRESH**
- Error counter threshold for error passive

Type Definitions

typedef `soc_periph_twai_clk_src_t twai_clock_source_t`
- RMT group clock source.

备注：User should select the clock source based on the power and resolution requirement

Enumerations

enum `twai_mode_t`
- TWAI Controller operating modes.
  - Values:

  enumerator **TWAI_MODE_NORMAL**
  - Normal operating mode where TWAI controller can send/receive/acknowledge messages

  enumerator **TWAI_MODE_NO_ACK**
  - Transmission does not require acknowledgment. Use this mode for self testing

  enumerator **TWAI_MODE_LISTEN_ONLY**
  - The TWAI controller will not influence the bus (No transmissions or acknowledgments) but can receive messages

Header File

- `components/driver/include/driver/twai.h`
**Functions**

\[ \texttt{esp_err_t twai_driver_install} \left( \text{const twai_general_config_t* } \text{g_config, const twai_timing_config_t* } \text{t_config, const twai_filter_config_t* } \text{f_config} \right) \]

Install TWAI driver.

This function installs the TWAI driver using three configuration structures. The required memory is allocated and the TWAI driver is placed in the stopped state after running this function.

**备注:** Macro initializers are available for the configuration structures (see documentation)

**备注:** To reinstall the TWAI driver, call `twai_driver_uninstall()` first

<table>
<thead>
<tr>
<th>参数</th>
<th>返回</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g_config</code> [in] General configuration structure</td>
<td>ESP_OK: Successfully installed TWAI driver</td>
</tr>
<tr>
<td><code>t_config</code> [in] Timing configuration structure</td>
<td>ESP_ERR_INVALID_ARG: Arguments are invalid, e.g. invalid clock source, invalid quanta resolution</td>
</tr>
<tr>
<td><code>f_config</code> [in] Filter configuration structure</td>
<td>ESP_ERR_NO_MEM: Insufficient memory</td>
</tr>
<tr>
<td></td>
<td>ESP_ERR_INVALID_STATE: Driver is already installed</td>
</tr>
</tbody>
</table>

\[ \texttt{esp_err_t twai_driver_uninstall} \left( \text{void} \right) \]

Uninstall the TWAI driver.

This function uninstalls the TWAI driver, freeing the memory utilized by the driver. This function can only be called when the driver is in the stopped state or the bus-off state.

**警告:** The application must ensure that no tasks are blocked on TX/RX queues or alerts when this function is called.

<table>
<thead>
<tr>
<th>返回</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_OK: Successfully uninstalled TWAI driver</td>
<td>ESP_OK: TWAI driver is now running</td>
</tr>
<tr>
<td>ESP_ERR_INVALID_STATE: Driver is not in stopped/bus-off state, or is not installed</td>
<td>ESP_ERR_INVALID_STATE: Driver is not in stopped state, or is not installed</td>
</tr>
</tbody>
</table>

\[ \texttt{esp_err_t twai_start} \left( \text{void} \right) \]

Start the TWAI driver.

This function starts the TWAI driver, putting the TWAI driver into the running state. This allows the TWAI driver to participate in TWAI bus activities such as transmitting/receiving messages. The TX and RX queue are reset in this function, clearing any messages that are unread or pending transmission. This function can only be called when the TWAI driver is in the stopped state.

<table>
<thead>
<tr>
<th>返回</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_OK: TWAI driver is now running</td>
<td>ESP_OK: TWAI driver is now running</td>
</tr>
<tr>
<td>ESP_ERR_INVALID_STATE: Driver is not in stopped state, or is not installed</td>
<td>ESP_ERR_INVALID_STATE: Driver is not in stopped state, or is not installed</td>
</tr>
</tbody>
</table>

\[ \texttt{esp_err_t twai_stop} \left( \text{void} \right) \]

Stop the TWAI driver.

This function stops the TWAI driver, preventing any further message from being transmitted or received until `twai_start()` is called. Any messages in the TX queue are cleared. Any messages in the RX queue should be read by the application after this function is called. This function can only be called when the TWAI driver is in the running state.
A message currently being transmitted/received on the TWAI bus will be ceased immediately. This may lead to other TWAI nodes interpreting the unfinished message as an error.

ESP_OK: TWAI driver is now Stopped
ESP_ERR_INVALID_STATE: Driver is not in running state, or is not installed

```c
esp_err_t twai_transmit (const twai_message_t *message, TickType_t ticks_to_wait)
```

Transmit a TWAI message.

This function queues a TWAI message for transmission. Transmission will start immediately if no other messages are queued for transmission. If the TX queue is full, this function will block until more space becomes available or until it times out. If the TX queue is disabled (TX queue length = 0 in configuration), this function will return immediately if another message is undergoing transmission. This function can only be called when the TWAI driver is in the running state and cannot be called under Listen Only Mode.

ESP_OK: Transmission successfully queued/initiated
ESP_ERR_INVALID_ARG: Arguments are invalid
ESP_ERR_TIMEOUT: Timed out waiting for space on TX queue
ESP_FAIL: TX queue is disabled and another message is currently transmitting
ESP_ERR_INVALID_STATE: TWAI driver is not in running state, or is not installed
ESP_ERR_NOT_SUPPORTED: Listen Only Mode does not support transmissions

```c
esp_err_t twai_receive (twai_message_t *message, TickType_t ticks_to_wait)
```

Receive a TWAI message.

This function receives a message from the RX queue. The flags field of the message structure will indicate the type of message received. This function will block if there are no messages in the RX queue.

ESP_OK: Message successfully received from RX queue
ESP_ERR_TIMEOUT: Timed out waiting for message
ESP_ERR_INVALID_ARG: Arguments are invalid
ESP_ERR_INVALID_STATE: TWAI driver is not installed

The flags field of the received message should be checked to determine if the received message contains any data bytes.

```
```

Parameters:

- *message* - [in] Message to transmit
- *ticks_to_wait* - [in] Number of FreeRTOS ticks to block on the TX queue

Returns:

- ESP_OK: Transmission successfully queued/initiated
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_TIMEOUT: Timed out waiting for space on TX queue
- ESP_FAIL: TX queue is disabled and another message is currently transmitting
- ESP_ERR_INVALID_STATE: TWAI driver is not in running state, or is not installed
- ESP_ERR_NOT_SUPPORTED: Listen Only Mode does not support transmissions

Parameters:

- *message* - [out] Received message
- *ticks_to_wait* - [in] Number of FreeRTOS ticks to block on RX queue

Returns:

- ESP_OK: Message successfully received from RX queue
- ESP_ERR_TIMEOUT: Timed out waiting for message
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed
esp_err_t twai_read_alerts(uint32_t*alerts, TickType_t ticks_to_wait)
Read TWAI driver alerts.

This function will read the alerts raised by the TWAI driver. If no alert has been issued when this function is called, this function will block until an alert occurs or until it timeouts.

备注：Multiple alerts can be raised simultaneously. The application should check for all alerts that have been enabled.

参数
- alerts [out] Bit field of raised alerts (see documentation for alert flags)
- ticks_to_wait [in] Number of FreeRTOS ticks to block for alert

返回
- ESP_OK: Alerts read
- ESP_ERR_TIMEOUT: Timed out waiting for alerts
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t twai_reconfigure_alerts(uint32_t alerts_enabled, uint32_t*current_alerts)
Reconfigure which alerts are enabled.

This function reconfigures which alerts are enabled. If there are alerts which have not been read whilst reconfiguring, this function can read those alerts.

参数
- alerts_enabled [in] Bit field of alerts to enable (see documentation for alert flags)
- current_alerts [out] Bit field of currently raised alerts. Set to NULL if unused

返回
- ESP_OK: Alerts reconfigured
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

esp_err_t twai_initiate_recovery(void)
Start the bus recovery process.

This function initiates the bus recovery process when the TWAI driver is in the bus-off state. Once initiated, the TWAI driver will enter the recovering state and wait for 128 occurrences of the bus-free signal on the TWAI bus before returning to the stopped state. This function will reset the TX queue, clearing any messages pending transmission.

备注：The BUS_RECOVERED alert can be enabled to alert the application when the bus recovery process completes.

返回
- ESP_OK: Bus recovery started
- ESP_ERR_INVALID_STATE: TWAI driver is not in the bus-off state, or is not installed

esp_err_t twai_get_status_info(twai_status_info_t*status_info)
Get current status information of the TWAI driver.

参数 status_info [out] Status information
返回
- ESP_OK: Status information retrieved
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: TWAI driver is not installed
### Chapter 2. API 参考

#### esp_err_t twai_clear_transmit_queue (void)
Clear the transmit queue.
This function will clear the transmit queue of all messages.

**备注：** The transmit queue is automatically cleared when twai_stop() or twai_initiate_recovery() is called.

**返回**
- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: TWAI driver is not installed or TX queue is disabled

#### esp_err_t twai_clear_receive_queue (void)
Clear the receive queue.
This function will clear the receive queue of all messages.

**备注：** The receive queue is automatically cleared when twai_start() is called.

**返回**
- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: TWAI driver is not installed

### Structures

**struct twai_general_config_t**
Structure for general configuration of the TWAI driver.

**备注：** Macro initializers are available for this structure

### Public Members

**twai_mode_t mode**
Mode of TWAI controller

**gpio_num_t tx_io**
Transmit GPIO number

**gpio_num_t rx_io**
Receive GPIO number

**gpio_num_t clkout_io**
CLKOUT GPIO number (optional, set to -1 if unused)

**gpio_num_t bus_off_io**
Bus off indicator GPIO number (optional, set to -1 if unused)

**uint32_t tx_queue_len**
Number of messages TX queue can hold (set to 0 to disable TX Queue)
uint32_t rx_queue_len
    Number of messages RX queue can hold

uint32_t alerts_enabled
    Bit field of alerts to enable (see documentation)

uint32_t clkout_divider
    CLKOUT divider. Can be 1 or any even number from 2 to 14 (optional, set to 0 if unused)

int intr_flags
    Interrupt flags to set the priority of the driver’s ISR. Note that to use the ESP_INTR_FLAG_IRAM, the CONFIG_TWAI_ISR_IN_IRAM option should be enabled first.

struct twai_status_info_t
    Structure to store status information of TWAI driver.

**Public Members**

twai_state_t state
    Current state of TWAI controller (Stopped/Running/Bus-Off/Recovery)

uint32_t msgs_to_tx
    Number of messages queued for transmission or awaiting transmission completion

uint32_t msgs_to_rx
    Number of messages in RX queue waiting to be read

uint32_t tx_error_counter
    Current value of Transmit Error Counter

uint32_t rx_error_counter
    Current value of Receive Error Counter

uint32_t tx_failed_count
    Number of messages that failed transmissions

uint32_t rx_missed_count
    Number of messages that were lost due to a full RX queue (or errata workaround if enabled)

uint32_t rx_overrun_count
    Number of messages that were lost due to a RX FIFO overrun

uint32_t arb_lost_count
    Number of instances arbitration was lost

uint32_t bus_error_count
    Number of instances a bus error has occurred
Chapter 2. API 参考

Macros

**TWAI_IO_UNUSED**

Marks GPIO as unused in TWAI configuration

Enumerations

```c
enum twai_state_t
```

TWAIdriver states.

**Values:**

- **enumerator TWAI_STATE_STOPPED**
  - Stopped state. The TWAI controller will not participate in any TWAI bus activities

- **enumerator TWAI_STATE_RUNNING**
  - Running state. The TWAI controller can transmit and receive messages

- **enumerator TWAI_STATE_BUS_OFF**
  - Bus-off state. The TWAI controller cannot participate in bus activities until it has recovered

- **enumerator TWAI_STATE_RECOVERING**
  - Recovering state. The TWAI controller is undergoing bus recovery

2.6.25 Universal Asynchronous Receiver/Transmitter (UART)

Overview

A Universal Asynchronous Receiver/Transmitter (UART) is a hardware feature that handles communication (i.e., timing requirements and data framing) using widely-adopted asynchronous serial communication interfaces, such as RS232, RS422, RS485. A UART provides a widely adopted and cheap method to realize full-duplex or half-duplex data exchange among different devices.

The ESP32 chip has three UART controllers (also referred to as port), each featuring an identical set of registers to simplify programming and for more flexibility.

Each UART controller is independently configurable with parameters such as baud rate, data bit length, bit ordering, number of stop bits, parity bit etc. All the controllers are compatible with UART-enabled devices from various manufacturers and can also support Infrared Data Association protocols (IrDA).

Functional Overview

The following overview describes how to establish communication between an ESP32 and other UART devices using the functions and data types of the UART driver. The overview reflects a typical programming workflow and is broken down into the sections provided below:

1. **Setting Communication Parameters** - Setting baud rate, data bits, stop bits, etc.
2. **Setting Communication Pins** - Assigning pins for connection to a device.
3. **Driver Installation** - Allocating ESP32’s resources for the UART driver.
4. **Running UART Communication** - Sending / receiving data
5. **Using Interrupts** - Triggering interrupts on specific communication events
6. **Deleting a Driver** - Freeing allocated resources if a UART communication is no longer required

Steps 1 to 3 comprise the configuration stage. Step 4 is where the UART starts operating. Steps 5 and 6 are optional. The UART driver’s functions identify each of the UART controllers using `uart_port_t`. This identification is needed for all the following function calls.
Setting Communication Parameters  UART communication parameters can be configured all in a single step or individually in multiple steps.

**Single Step**  Call the function `uart_param_config()` and pass to it a `uart_config_t` structure. The `uart_config_t` structure should contain all the required parameters. See the example below.

```c
const uart_port_t uart_num = UART_NUM_2;
uart_config_t uart_config = {
    .baud_rate = 115200,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_CTS_RTS,
    .rx_flow_ctrl_thresh = 122,
};
// Configure UART parameters
ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
```

For more information on how to configure the hardware flow control options, please refer to `peripherals/uart/uart_echo`.

**Multiple Steps**  Configure specific parameters individually by calling a dedicated function from the table given below. These functions are also useful if re-configuring a single parameter.

<table>
<thead>
<tr>
<th>Parameter to Configure</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud rate</td>
<td><code>uart_set_baudrate()</code></td>
</tr>
<tr>
<td>Number of transmitted bits</td>
<td><code>uart_set_word_length()</code> selected out of <code>uart_word_length_t</code></td>
</tr>
<tr>
<td>Parity control</td>
<td><code>uart_set_parity()</code> selected out of <code>uart_parity_t</code></td>
</tr>
<tr>
<td>Number of stop bits</td>
<td><code>uart_set_stop_bits()</code> selected out of <code>uart_stop_bits_t</code></td>
</tr>
<tr>
<td>Hardware flow control mode</td>
<td><code>uart_set_hw_flow_ctrl()</code> selected out of <code>uart_hw_flowcontrol_t</code></td>
</tr>
<tr>
<td>Communication mode</td>
<td><code>uart_set_mode()</code> selected out of <code>uart_mode_t</code></td>
</tr>
</tbody>
</table>

Each of the above functions has a `_get_` counterpart to check the currently set value. For example, to check the current baud rate value, call `uart_get_baudrate()`.

Setting Communication Pins  After setting communication parameters, configure the physical GPIO pins to which the other UART device will be connected. For this, call the function `uart_set_pin()` and specify the GPIO pin numbers to which the driver should route the Tx, Rx, RTS, and CTS signals. If you want to keep a currently allocated pin number for a specific signal, pass the macro `UART_PIN_NO_CHANGE`.

The same macro should be specified for pins that will not be used.

```c
// Set UART pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)
ESP_ERROR_CHECK(uart_set_pin(UART_NUM_2, 4, 5, 18, 19));
```

Driver Installation  Once the communication pins are set, install the driver by calling `uart_driver_install()` and specify the following parameters:

- Size of Tx ring buffer
- Size of Rx ring buffer
- Event queue handle and size
- Flags to allocate an interrupt

The function will allocate the required internal resources for the UART driver.
**Chapter 2. API 参考**

```c
// Setup UART buffered IO with event queue
const int uart_buffer_size = (1024 * 2);
QueueHandle_t uart_queue;
// Install UART driver using an event queue here
ESP_ERROR_CHECK(uart_driver_install(UART_NUM_2, uart_buffer_size,
        uart_buffer_size, 10, &uart_queue, 0));
```

Once this step is complete, you can connect the external UART device and check the communication.

### Running UART Communication

Serial communication is controlled by each UART controller’s finite state machine (FSM).

The process of sending data involves the following steps:

1. Write data into Tx FIFO buffer
2. FSM serializes the data
3. FSM sends the data out

The process of receiving data is similar, but the steps are reversed:

1. FSM processes an incoming serial stream and parallelizes it
2. FSM writes the data into Rx FIFO buffer
3. Read the data from Rx FIFO buffer

Therefore, an application will be limited to writing and reading data from a respective buffer using `uart_write_bytes()` and `uart_read_bytes()` respectively, and the FSM will do the rest.

### Transmitting

After preparing the data for transmission, call the function `uart_write_bytes()` and pass the data buffer’s address and data length to it. The function will copy the data to the Tx ring buffer (either immediately or after enough space is available), and then exit. When there is free space in the Tx FIFO buffer, an interrupt service routine (ISR) moves the data from the Tx ring buffer to the Tx FIFO buffer in the background. The code below demonstrates the use of this function.

```c
// Write data to UART.
char* test_str = "This is a test string\n";
uart_write_bytes(uart_num, (const char*)test_str, strlen(test_str));
```

The function `uart_write_bytes_with_break()` is similar to `uart_write_bytes()` but adds a serial break signal at the end of the transmission. A ‘serial break signal’ means holding the Tx line low for a period longer than one data frame.

```c
// Write data to UART, end with a break signal.
uart_write_bytes_with_break(uart_num, "test break\n", strlen("test break\n"), 100);
```

Another function for writing data to the Tx FIFO buffer is `uart_tx_chars()`. Unlike `uart_write_bytes()`, this function will not block until space is available. Instead, it will write all data which can immediately fit into the hardware Tx FIFO, and then return the number of bytes that were written.

There is a ‘companion’ function `uart_waitTxDone()` that monitors the status of the Tx FIFO buffer and returns once it is empty.

```c
// Wait for packet to be sent
const uart_port_t uart_num = UART_NUM_2;
ESP_ERROR_CHECK(uart_wait_tx_done(uart_num, 100)); // wait timeout is 100 RTOS_ticks (TickType_t)
```

### Receiving

Once the data is received by the UART and saved in the Rx FIFO buffer, it needs to be retrieved using the function `uart_read_bytes()`. Before reading data, you can check the number of bytes available in the Rx FIFO buffer by calling `uart_get_buffered_data_len()`. An example of using these functions is given below.
// Read data from UART.
const uart_port_t uart_num = UART_NUM_2;
uint8_t data[128];
int length = 0;
ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num,(size_t*)&length));
length = uart_read_bytes(uart_num, data, length, 100);

If the data in the Rx FIFO buffer is no longer needed, you can clear the buffer by calling `uart_flush()`.

### Software Flow Control
If the hardware flow control is disabled, you can manually set the RTS and DTR signal levels by using the functions `uart_set_rts()` and `uart_set_dtr()` respectively.

### Communication Mode Selection
The UART controller supports a number of communication modes. A mode can be selected using the function `uart_set_mode()`. Once a specific mode is selected, the UART driver will handle the behavior of a connected UART device accordingly. As an example, it can control the RS485 driver chip using the RTS line to allow half-duplex RS485 communication.

// Setup UART in rs485 half duplex mode
ESP_ERROR_CHECK(uart_set_mode(uart_num, UART_MODE_RS485_HALF_DUPLEX));

### Using Interrupts
There are many interrupts that can be generated following specific UART states or detected errors. The full list of available interrupts is provided in ESP32 Technical Reference Manual > UART Controller (UART) > UART Interrupts and UHCI Interrupts [PDF]. You can enable or disable specific interrupts by calling `uart_enable_intr_mask()` or `uart_disable_intr_mask()` respectively.

The `uart_driver_install()` function installs the driver’s internal interrupt handler to manage the Tx and Rx ring buffers and provides high-level API functions like events (see below).

The API provides a convenient way to handle specific interrupts discussed in this document by wrapping them into dedicated functions:

- **Event detection**: There are several events defined in `uart_event_type_t` that may be reported to a user application using the FreeRTOS queue functionality. You can enable this functionality when calling `uart_driver_install()` described in Driver Installation. An example of using Event detection can be found in peripherals/uart/uart_events.
  - **FIFO space threshold or transmission timeout reached**: The Tx and Rx FIFO buffers can trigger an interrupt when they are filled with a specific number of characters, or on a timeout of sending or receiving data. To use these interrupts, do the following:
    - Configure respective threshold values of the buffer length and timeout by entering them in the structure `uart_intr_config_t` and calling `uart_intr_config()`
    - Enable the interrupts using the functions `uart_enable_tx_intr()` and `uart_enable_rx_intr()`
    - Disable these interrupts using the corresponding functions `uart_disable_tx_intr()` or `uart_disable_rx_intr()`

- **Pattern detection**: An interrupt triggered on detecting a ‘pattern’ of the same character being received/sent repeatedly for a number of times. This functionality is demonstrated in the example peripherals/uart/uart_events. It can be used, e.g., to detect a command string followed by a specific number of identical characters (the ‘pattern’) added at the end of the command string. The following functions are available:
    - Configure and enable this interrupt using `uart_enable_pattern_det_intr()`
    - Disable the interrupt using `uart_disable_pattern_det_intr()`

### Macros
The API also defines several macros. For example, `UART_FIFO_LEN` defines the length of hardware FIFO buffers; `UART_BITRATE_MAX` gives the maximum baud rate supported by the UART controllers, etc.

### Deleting a Driver
If the communication established with `uart_driver_install()` is no longer required, the driver can be removed to free allocated resources by calling `uart_driver_delete()`.
Overview of RS485 specific communication options

The following section will use `[UART_REGISTER_NAME].[UART_FIELD_BIT]` to refer to UART register fields/bits. For more information on a specific option bit, see ESP32 Technical Reference Manual > UART Controller (UART) > Register Summary [PDF]. Use the register name to navigate to the register description and then find the field/bit.

- UART_RS485_CONF_REG.UART_RS485_EN: setting this bit enables RS485 communication mode support.
- UART_RS485_CONF_REG.UART_RS485_TX_RX_EN: if this bit is set, the transmitter’s output signal loops back to the receiver’s input signal.
- UART_RS485_CONF_REG.UART_RS485_RXBY_TX_EN: if this bit is set, the transmitter will still be sending data if the receiver is busy (remove collisions automatically by hardware).

The ESP32’s RS485 UART hardware can detect signal collisions during transmission of a datagram and generate the interrupt UART_RS485_CLASH_INT if this interrupt is enabled. The term collision means that a transmitted datagram is not equal to the one received on the other end. Data collisions are usually associated with the presence of other active devices on the bus or might occur due to bus errors.

The collision detection feature allows handling collisions when their interrupts are activated and triggered. The interrupts UART_RS485_FRM_ERR_INT and UART_RS485_PARITY_ERR_INT can be used with the collision detection feature to control frame errors and parity bit errors accordingly in RS485 mode. This functionality is supported in the UART driver and can be used by selecting the UART_MODE_RS485_APP_CTRL mode (see the function `uart_set_mode()`).

The collision detection feature can work with circuit A and circuit C (see Section Interface Connection Options). In the case of using circuit A or B, the RTS pin connected to the DE pin of the bus driver should be controlled by the user application. Use the function `uart_get_collision_flag()` to check if the collision detection flag has been raised.

The ESP32 UART controllers themselves do not support half-duplex communication as they cannot provide automatic control of the RTS pin connected to the -RE/DE input of RS485 bus driver. However, half-duplex communication can be achieved via software control of the RTS pin by the UART driver. This can be enabled by selecting the UART_MODE_RS485_HALF_DUPLEX mode when calling `uart_set_mode()`.

Once the host starts writing data to the Tx FIFO buffer, the UART driver automatically asserts the RTS pin (logic 1); once the last bit of the data has been transmitted, the driver de-asserts the RTS pin (logic 0). To use this mode, the software would have to disable the hardware flow control function. This mode works with all the used circuits shown below.

Interface Connection Options  This section provides example schematics to demonstrate the basic aspects of ESP32’s RS485 interface connection.

备注:  The schematics below do not necessarily contain all required elements.

- The analog devices ADM483 & ADM2483 are examples of common RS485 transceivers and can be replaced with other similar transceivers.

Circuit A: Collision Detection Circuit

![Circuit A schematic diagram]

(下页续)
This circuit is preferable because it allows for collision detection and is quite simple at the same time. The receiver in the line driver is constantly enabled, which allows the UART to monitor the RS485 bus. Echo suppression is performed by the UART peripheral when the bit UART_RS485_CONF_REG.UART_RS485TX_RX_EN is enabled.

**Circuit B: Manual Switching Transmitter/Receiver Without Collision Detection**

This circuit does not allow for collision detection. It suppresses the null bytes that the hardware receives when the bit UART_RS485_CONF_REG.UART_RS485TX_RX_EN is set. The bit UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN is not applicable in this case.

**Circuit C: Auto Switching Transmitter/Receiver**

This galvanically isolated circuit does not require RTS pin control by a software application or driver because it controls the transceiver direction automatically. However, it requires suppressing null bytes during transmission by setting UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN to 1 and UART_RS485_CONF_REG.UART_RS485TX_RX_EN to 0. This setup can work in any RS485 UART mode or even in UART_MODE_UART.
Application Examples

The table below describes the code examples available in the directory `peripherals/uart/`.

<table>
<thead>
<tr>
<th>Code Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peripherals/uart/uart_echo</td>
<td>Configuring UART settings, installing the UART driver, and reading/writing over the UART1 interface.</td>
</tr>
<tr>
<td>peripherals/uart/uart_events</td>
<td>Reporting various communication events, using pattern detection interrupts.</td>
</tr>
<tr>
<td>peripherals/uart/uart_async_rxtxtasks</td>
<td>Transmitting and receiving data in two separate FreeRTOS tasks over the same UART.</td>
</tr>
<tr>
<td>peripherals/uart/uart_select</td>
<td>Using synchronous I/O multiplexing for UART file descriptors.</td>
</tr>
<tr>
<td>peripherals/uart/uart_echo_rs485</td>
<td>Setting up UART driver to communicate over RS485 interface in half-duplex mode. This example is similar to <code>peripherals/uart/uart_echo</code> but allows communication through an RS485 interface chip connected to ESP32 pins.</td>
</tr>
<tr>
<td>peripherals/uart/nmea0183_parser</td>
<td>Obtaining GPS information by parsing NMEA0183 statements received from GPS via the UART peripheral.</td>
</tr>
</tbody>
</table>

API Reference

Header File

- components/driver/include/driver/uart.h

Functions

```c
esp_err_t uart_driver_install (uart_port_t uart_num, int rx_buffer_size, int tx_buffer_size, int queue_size, QueueHandle_t *uart_queue, int intr_alloc_flags)
```

Install UART driver and set the UART to the default configuration.

UART ISR handler will be attached to the same CPU core that this function is running on.

**备注:** Rx_buffer_size should be greater than UART_FIFO_LEN. Tx_buffer_size should be either zero or greater than UART_FIFO_LEN.

参数

- **uart_num** – UART port number, the max port number is (UART_NUM_MAX -1).
- **rx_buffer_size** – UART RX ring buffer size.
- **tx_buffer_size** – UART TX ring buffer size. If set to zero, driver will not use TX buffer, TX function will block task until all data have been sent out.
- **queue_size** – UART event queue size/depth.
- **uart_queue** – UART event queue handle (out param). On success, a new queue handle is written here to provide access to UART events. If set to NULL, driver will not use an event queue.
- **intr_alloc_flags** – Flags used to allocate the interrupt. One or multiple (ORed) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info. Do not set ESP_INTR_FLAG_IRAM here (the driver’s ISR handler is not located in IRAM)

返回

- ESP_OK Success
- ESP_FAIL Parameter error

```c
esp_err_t uart_driver_delete (uart_port_t uart_num)
```

Uninstall UART driver.

参数 **uart_num** – UART port number, the max port number is (UART_NUM_MAX -1).

返回
Chapter 2. API 参考

- ESP_OK Success
- ESP_FAIL Parameter error

`bool uart_is_driver_installed(uart_port_t uart_num)`
Checks whether the driver is installed or not.

参数 `uart_num` – UART port number, the max port number is (UART_NUM_MAX - 1).

返回
- true driver is installed
- false driver is not installed

`esp_err_t uart_set_word_length(uart_port_t uart_num, uart_word_length_t data_bit)`
Set UART data bits.

参数
- `uart_num` – UART port number, the max port number is (UART_NUM_MAX - 1).
- `data_bit` – UART data bits

返回
- ESP_OK Success
- ESP_FAIL Parameter error

`esp_err_t uart_get_word_length(uart_port_t uart_num, uart_word_length_t *data_bit)`
Get the UART data bit configuration.

参数
- `uart_num` – UART port number, the max port number is (UART_NUM_MAX - 1).
- `data_bit` – Pointer to accept value of UART data bits.

返回
- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*data_bit)

`esp_err_t uart_set_stop_bits(uart_port_t uart_num, uart_stop_bits_t stop_bits)`
Set UART stop bits.

参数
- `uart_num` – UART port number, the max port number is (UART_NUM_MAX - 1).
- `stop_bits` – UART stop bits

返回
- ESP_OK Success
- ESP_FAIL Fail

`esp_err_t uart_get_stop_bits(uart_port_t uart_num, uart_stop_bits_t *stop_bits)`
Get the UART stop bit configuration.

参数
- `uart_num` – UART port number, the max port number is (UART_NUM_MAX - 1).
- `stop_bits` – Pointer to accept value of UART stop bits.

返回
- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*stop_bits)

`esp_err_t uart_set_parity(uart_port_t uart_num, uart_parity_t parity_mode)`
Set UART parity mode.

参数
- `uart_num` – UART port number, the max port number is (UART_NUM_MAX - 1).
- `parity_mode` – the enum of uart parity configuration

返回
- ESP_FAIL Parameter error
- ESP_OK Success

`esp_err_t uart_get_parity(uart_port_t uart_num, uart_parity_t *parity_mode)`
Get the UART parity mode configuration.
Chapter 2. API 参考

参数
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).
- parity_mode - Pointer to accept value of UART parity mode.

返回
- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*parity_mode)

```c
esp_err_t uart_get_sclk_freq(uart_sclk_t sclk, uint32_t*out_freq_hz)
```

Get the frequency of a clock source for the UART.

参数
- sclk - Clock source
- out_freq_hz - [out] Output of frequency, in Hz

返回
- ESP_ERR_INVALID_ARG: if the clock source is not supported
- otherwise ESP_OK

```c
esp_err_t uart_set_baudrate(uart_port_t uart_num, uint32_t baudrate)
```

Set UART baud rate.

参数
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).
- baudrate - UART baud rate.

返回
- ESP_FAIL Parameter error
- ESP_OK Success

```c
esp_err_t uart_get_baudrate(uart_port_t uart_num, uint32_t*baudrate)
```

Get the UART baud rate configuration.

参数
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).
- baudrate - Pointer to accept value of UART baud rate

返回
- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*baudrate)

```c
esp_err_t uart_set_line_inverse(uart_port_t uart_num, uint32_t inverse_mask)
```

Set UART line inverse mode.

参数
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).
- inverse_mask - Choose the wires that need to be inverted. Using the ORred mask of uart_signal_inv_t

返回
- ESP_OK Success
- ESP_FAIL Parameter error

```c
esp_err_t uart_set_hw_flow_ctrl(uart_port_t uart_num, uart_hw_flowcontrol_t flow_ctrl, uint8_t rx_thresh)
```

Set hardware flow control.

参数
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).
- flow_ctrl - Hardware flow control mode
- rx_thresh - Threshold of Hardware RX flow control (0 ~ UART_FIFO_LEN). Only when UART_HW_FLOWCTRL_RTS is set, will the rx_thresh value be set.

返回
- ESP_OK Success
- ESP_FAIL Parameter error
**esp_err_t uart_set_sw_flow_ctrl** (*uart_port_t* *uart_num*, bool *enable*, uint8_t *rx_thresh_xon*, uint8_t *rx_thresh_xoff*)

Set software flow control.

- **uart_num** - UART NUM_0, UART NUM_1 or UART NUM_2
- **enable** - switch on or off
- **rx_thresh_xon** - low water mark
- **rx_thresh_xoff** - high water mark

**Return**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_get_hw_flow_ctrl** (*uart_port_t* *uart_num*, *uart_hw_flowcontrol_t* *flow_ctrl*)

Get the UART hardware flow control configuration.

- **uart_num** - UART port number, the max port number is (UART_NUM_MAX -1).
- **flow_ctrl** - Option for different flow control mode.

**Return**
- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*flow_ctrl*

**esp_err_t uart_clear_intr_status** (*uart_port_t* *uart_num*, uint32_t *clr_mask*)

Clear UART interrupt status.

- **uart_num** - UART port number, the max port number is (UART_NUM_MAX -1).
- **clr_mask** - Bit mask of the interrupt status to be cleared.

**Return**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_enable_intr_mask** (*uart_port_t* *uart_num*, uint32_t *enable_mask*)

Set UART interrupt enable.

- **uart_num** - UART port number, the max port number is (UART_NUM_MAX -1).
- **enable_mask** - Bit mask of the enable bits.

**Return**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_disable_intr_mask** (*uart_port_t* *uart_num*, uint32_t *disable_mask*)

Clear UART interrupt enable bits.

- **uart_num** - UART port number, the max port number is (UART_NUM_MAX -1).
- **disable_mask** - Bit mask of the disable bits.

**Return**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_enable_rx_intr** (*uart_port_t* *uart_num*)

Enable UART RX interrupt (RX_FULL & RX_TIMEOUT INTERRUPT)

- **uart_num** - UART port number, the max port number is (UART_NUM_MAX -1).

**Return**
- ESP_OK Success
- ESP_FAIL Parameter error
**esp_err_t uart_disable_rx_intr (uart_port_t uart_num)**
Disable UART RX interrupt (RX_FULL & RX_TIMEOUT_INTERRUPT)

**参数**
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_disable_tx_intr (uart_port_t uart_num)**
Disable UART TX interrupt (TX_FULL & TX_TIMEOUT_INTERRUPT)

**参数**
- uart_num - UART port number

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_enable_tx_intr (uart_port_t uart_num, int enable, int thresh)**
Enable UART TX interrupt (TX_FULL & TX_TIMEOUT_INTERRUPT)

**参数**
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).
- enable - 1: enable; 0: disable
- thresh - Threshold of TX interrupt, 0 ~ UART_FIFO_LEN

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_set_pin (uart_port_t uart_num, int tx_io_num, int rx_io_num, int rts_io_num, int cts_io_num)**
Assign signals of a UART peripheral to GPIO pins.

**备注:** If the GPIO number configured for a UART signal matches one of the IOMUX signals for that GPIO, the signal will be connected directly via the IOMUX. Otherwise the GPIO and signal will be connected via the GPIO Matrix. For example, if on an ESP32 the call `uart_set_pin(0, 1, 3, -1, -1)` is performed, as GPIO1 is UART0’s default TX pin and GPIO3 is UART0’s default RX pin, both will be connected to respectively U0TXD and U0RXD through the IOMUX, totally bypassing the GPIO matrix. The check is performed on a per-pin basis. Thus, it is possible to have RX pin binded to a GPIO through the GPIO matrix, whereas TX is binded to its GPIO through the IOMUX.

**备注:** Internal signal can be output to multiple GPIO pads. Only one GPIO pad can connect with input signal.

**参数**
- uart_num - UART port number, the max port number is (UART_NUM_MAX -1).
- tx_io_num - UART TX pin GPIO number.
- rx_io_num - UART RX pin GPIO number.
- rts_io_num - UART RTS pin GPIO number.
- cts_io_num - UART CTS pin GPIO number.

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_set_rts (uart_port_t uart_num, int level)**
Manually set the UART RTS pin level.

**备注:** UART must be configured with hardware flow control disabled.
## Chapter 2. API 参考

### 参数
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `level` - 1: RTS output low (active); 0: RTS output high (block)

### 返回
- ESP_OK Success
- ESP_FAIL Parameter error

```c
esp_err_t esp_set_dtr (uart_port_t uart_num, int level)
```
Manually set the UART DTR pin level.

**参数**
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `level` - 1: DTR output low; 0: DTR output high

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

```c
esp_err_t esp_set_tx_idle_num (uart_port_t uart_num, uint16_t idle_num)
```
Set UART idle interval after tx FIFO is empty.

**参数**
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `idle_num` - idle interval after tx FIFO is empty(unit: the time it takes to send one bit under current baudrate)

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

```c
esp_err_t esp_param_config (uart_port_t uart_num, const uart_config_t *uart_config)
```
Set UART configuration parameters.

**参数**
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `uart_config` - UART parameter settings

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

```c
esp_err_t esp_intr_config (uart_port_t uart_num, const uart_intr_config_t *intr_conf)
```
Configure UART interrupts.

**参数**
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `intr_conf` - UART interrupt settings

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error

```c
esp_err_t esp_wait_tx_done (uart_port_t uart_num, TickType_t ticks_to_wait)
```
Wait until UART TX FIFO is empty.

**参数**
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `ticks_to_wait` - Timeout, count in RTOS ticks

**返回**
- ESP_OK Success
- ESP_FAIL Parameter error
- ESP_ERR_TIMEOUT Timeout

```c
int uart_tx_chars (uart_port_t uart_num, const char *buffer, uint32_t len)
```
Send data to the UART port from a given buffer and length.
This function will not wait for enough space in TX FIFO. It will just fill the available TX FIFO and return when the FIFO is full.

备注: This function should only be used when UART TX buffer is not enabled.

### Parameters
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `buffer` - data buffer address
- `len` - data length to send

### Return
- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

```c
int uart_write_bytes (uart_port_t uart_num, const void *src, size_t size)
```

Send data to the UART port from a given buffer and length.

If the UART driver’s parameter ‘tx_buffer_size’ is set to zero: This function will not return until all the data have been sent out, or at least pushed into TX FIFO.

Otherwise, if the ‘tx_buffer_size’ > 0, this function will return after copying all the data to tx ring buffer, UART ISR will then move data from the ring buffer to TX FIFO gradually.

### Parameters
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `src` - data buffer address
- `size` - data length to send

### Return
- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

```c
int uart_write_bytes_with_break (uart_port_t uart_num, const void *src, size_t size, int brk_len)
```

Send data to the UART port from a given buffer and length.

If the UART driver’s parameter ‘tx_buffer_size’ is set to zero: This function will not return until all the data and the break signal have been sent out. After all data is sent out, send a break signal.

Otherwise, if the ‘tx_buffer_size’ > 0, this function will return after copying all the data to tx ring buffer, UART ISR will then move data from the ring buffer to TX FIFO gradually. After all data sent out, send a break signal.

### Parameters
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `src` - data buffer address
- `size` - data length to send
- `brk_len` - break signal duration(unit: the time it takes to send one bit at current baudrate)

### Return
- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

```c
int uart_read_bytes (uart_port_t uart_num, void *buf, uint32_t length, TickType_t ticks_to_wait)
```

UART read bytes from UART buffer.

### Parameters
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `buf` - pointer to the buffer.
- `length` - data length
- `ticks_to_wait` - s Timeout, count in RTOS ticks

### Return
- (-1) Error
- OTHERS (>=0) The number of bytes read from UART buffer
**esp_err_t uart_flush (uart_port_t uart_num)**

Alias of `uart_flush_input`. UART ring buffer flush. This will discard all data in the UART RX buffer.

**备注:** Instead of waiting the data sent out, this function will clear UART rx buffer. In order to send all the data in tx FIFO, we can use `uart_wait_tx_done` function.

**参数:** `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).

**返回:**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_flush_input (uart_port_t uart_num)**

Clear input buffer, discard all the data is in the ring-buffer.

**备注:** In order to send all the data in tx FIFO, we can use `uart_wait_tx_done` function.

**参数:** `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).

**返回:**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_get_buffered_data_len (uart_port_t uart_num, size_t *size)**

UART get RX ring buffer cached data length.

**参数:**
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `size` - Pointer of size_t to accept cached data length

**返回:**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_get_tx_buffer_free_size (uart_port_t uart_num, size_t *size)**

UART get TX ring buffer free space size.

**参数:**
- `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).
- `size` - Pointer of size_t to accept the free space size

**返回:**
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

**esp_err_t uart_disable_pattern_det_intr (uart_port_t uart_num)**

UART disable pattern detect function. Designed for applications like ‘AT commands’. When the hardware detects a series of one same character, the interrupt will be triggered.

**参数:** `uart_num` - UART port number, the max port number is (UART_NUM_MAX -1).

**返回:**
- ESP_OK Success
- ESP_FAIL Parameter error

**esp_err_t uart_enable_pattern_det_baud_intr (uart_port_t uart_num, char pattern_chr, uint8_t chr_num, int chr_tout, int post_idle, int pre_idle)**

UART enable pattern detect function. Designed for applications like ‘AT commands’. When the hardware detect a series of one same character, the interrupt will be triggered.

**参数:** `uart_num` - UART port number.
Chapter 2. API

- **pattern_chr** – character of the pattern.
- **chr_num** – number of the character, 8-bit value.
- **chr_tout** – timeout of the interval between each pattern characters, 16-bit value, unit is the baud-rate cycle you configured. When the duration is more than this value, it will not take this data as at_cmd char.
- **post_idle** – idle time after the last pattern character, 16-bit value, unit is the baud-rate cycle you configured. When the duration is less than this value, it will not take the previous data as the last at_cmd char
- **pre_idle** – idle time before the first pattern character, 16-bit value, unit is the baud-rate cycle you configured. When the duration is less than this value, it will not take this data as the first at_cmd char.

返回

- ESP_OK Success
- ESP_FAIL Parameter error

```c
int uart_pattern_pop_pos(uart_port_t uart_num)
```

Return the nearest detected pattern position in buffer. The positions of the detected pattern are saved in a queue, this function will dequeue the first pattern position and move the pointer to next pattern position.

The following APIs will modify the pattern position info: uart_flush_input, uart_read_bytes, uart_driver_delete, uart_pop_pattern_pos It is the application’s responsibility to ensure atomic access to the pattern queue and the rx data buffer when using pattern detect feature.

备注：If the RX buffer is full and flow control is not enabled, the detected pattern may not be found in the rx buffer due to overflow.

```c
int uart_pattern_get_pos(uart_port_t uart_num)
```

Return the nearest detected pattern position in buffer. The positions of the detected pattern are saved in a queue, This function do nothing to the queue.

The following APIs will modify the pattern position info: uart_flush_input, uart_read_bytes, uart_driver_delete, uart_pop_pattern_pos It is the application’s responsibility to ensure atomic access to the pattern queue and the rx data buffer when using pattern detect feature.

备注：If the RX buffer is full and flow control is not enabled, the detected pattern may not be found in the rx buffer due to overflow.

```c
esp_err_t uart_pattern_queue_reset(uart_port_t uart_num, int queue_length)
```

Allocate a new memory with the given length to save record the detected pattern position in rx buffer.

参数

- **uart_num** – UART port number, the max port number is (UART_NUM_MAX -1).

返回

- (-1) No pattern found for current index or parameter error
- others the pattern position in rx buffer.
Chapter 2. API 参考

- **queue_length** - Max queue length for the detected pattern. If the queue length is not large enough, some pattern positions might be lost. Set this value to the maximum number of patterns that could be saved in data buffer at the same time.

返回
- ESP_ERR_NO_MEM No enough memory
- ESP_ERR_INVALID_STATE Driver not installed
- ESP_FAIL Parameter error
- ESP_OK Success

```c
esp_err_t uart_set_mode(uart_port_t uart_num, uart_mode_t mode)
```
UART set communication mode.

备注：This function must be executed after uart_driver_install(), when the driver object is initialized.

参数
- **uart_num** - Uart number to configure, the max port number is (UART_NUM_MAX -1).
- **mode** - UART UART mode to set

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

```c
esp_err_t uart_set_rx_full_threshold(uart_port_t uart_num, int threshold)
```
Set uart threshold value for RX fifo full.

备注：If application is using higher baudrate and it is observed that bytes in hardware RX fifo are overwritten then this threshold can be reduced

参数
- **uart_num** - UART_NUM_0, UART_NUM_1 or UART_NUM_2
- **threshold** - Threshold value above which RX fifo full interrupt is generated

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

```c
esp_err_t uart_set_tx_empty_threshold(uart_port_t uart_num, int threshold)
```
Set uart threshold values for TX fifo empty.

参数
- **uart_num** - UART_NUM_0, UART_NUM_1 or UART_NUM_2
- **threshold** - Threshold value below which TX fifo empty interrupt is generated

返回
- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

```c
esp_err_t uart_set_rx_timeout(uart_port_t uart_num, const uint8_t tout_thresh)
```
UART set threshold timeout for TOUT feature.

参数
- **uart_num** - Uart number to configure, the max port number is (UART_NUM_MAX -1).
- **tout_thresh** - This parameter defines timeout threshold in uart symbol periods. The maximum value of threshold is 126. tout_thresh = 1, defines TOUT interrupt timeout equal to transmission time of one symbol (~11 bit) on current baudrate. If the time is
expired the UART_RXFIFO_TOUT_INT interrupt is triggered. If tout_thresh == 0, the TOUT feature is disabled.

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

**esp_err_t uart_get_collision_flag(uart_port_t uart_num, bool *collision_flag)**

Returns collision detection flag for RS485 mode. Function returns the collision detection flag into variable pointed by collision_flag. *collision_flag = true, if collision detected else it is equal to false. This function should be executed when actual transmission is completed (after uart_write_bytes()).

- **uart_num** - Uart number to configure the max port number is (UART_NUM_MAX -1).
- **collision_flag** - Pointer to variable of type bool to return collision flag.

**esp_err_t uart_set_wakeup_threshold(uart_port_t uart_num, int wakeup_threshold)**

Set the number of RX pin signal edges for light sleep wakeup.

UART can be used to wake up the system from light sleep. This feature works by counting the number of positive edges on RX pin and comparing the count to the threshold. When the count exceeds the threshold, system is woken up from light sleep. This function allows setting the threshold value.

Stop bit and parity bits (if enabled) also contribute to the number of edges. For example, letter ‘a’ with ASCII code 97 is encoded as 0100001101 on the wire (with 8n1 configuration), start and stop bits included. This sequence has 3 positive edges (transitions from 0 to 1). Therefore, to wake up the system when ‘a’ is sent, set wakeup_threshold=3.

The character that triggers wakeup is not received by UART (i.e. it can not be obtained from UART FIFO). Depending on the baud rate, a few characters after that will also not be received. Note that when the chip enters and exits light sleep mode, APB frequency will be changing. To make sure that UART has correct baud rate all the time, select UART_SCLK_REF_TICK or UART_SCLK_XTAL as UART clock source in `uart_config_t::source_clk`.

**备注：** in ESP32, the wakeup signal can only be input via IO_MUX (i.e. GPIO3 should be configured as function_1 to wake up UART0, GPIO9 should be configured as function_5 to wake up UART1), UART2 does not support light sleep wakeup feature.

- **uart_num** - UART number, the max port number is (UART_NUM_MAX -1).
- **wakeup_threshold** - number of RX edges for light sleep wakeup, value is 3 .. 0x3ff.

**esp_err_t uart_get_wakeup_threshold(uart_port_t uart_num, int *out_wakeup_threshold)**

Get the number of RX pin signal edges for light sleep wakeup.

See description of uart_set_wakeup_threshold for the explanation of UART wakeup feature.

- **uart_num** - UART number, the max port number is (UART_NUM_MAX -1).
- **out_wakeup_threshold** - [out] output, set to the current value of wakeup threshold for the given UART.

**返回**
- ESP_OK on success
- ESP_ERR_INVALID_ARG if uart_num is incorrect or wakeup_threshold is outside of [3, 0x3ff] range.

**esp_err_t uart_set_wakeup_threshold(uart_port_t uart_num, int wakeup_threshold)**

Set the number of RX pin signal edges for light sleep wakeup.

UART can be used to wake up the system from light sleep. This feature works by counting the number of positive edges on RX pin and comparing the count to the threshold. When the count exceeds the threshold, system is woken up from light sleep. This function allows setting the threshold value.

Stop bit and parity bits (if enabled) also contribute to the number of edges. For example, letter ‘a’ with ASCII code 97 is encoded as 0100001101 on the wire (with 8n1 configuration), start and stop bits included. This sequence has 3 positive edges (transitions from 0 to 1). Therefore, to wake up the system when ‘a’ is sent, set wakeup_threshold=3.

The character that triggers wakeup is not received by UART (i.e. it can not be obtained from UART FIFO). Depending on the baud rate, a few characters after that will also not be received. Note that when the chip enters and exits light sleep mode, APB frequency will be changing. To make sure that UART has correct baud rate all the time, select UART_SCLK_REF_TICK or UART_SCLK_XTAL as UART clock source in `uart_config_t::source_clk`.

**备注：** in ESP32, the wakeup signal can only be input via IO_MUX (i.e. GPIO3 should be configured as function_1 to wake up UART0, GPIO9 should be configured as function_5 to wake up UART1), UART2 does not support light sleep wakeup feature.

- **uart_num** - UART number, the max port number is (UART_NUM_MAX -1).
- **wakeup_threshold** - number of RX edges for light sleep wakeup, value is 3 .. 0x3ff.

**esp_err_t uart_get_wakeup_threshold(uart_port_t uart_num, int *out_wakeup_threshold)**

Get the number of RX pin signal edges for light sleep wakeup.

See description of uart_set_wakeup_threshold for the explanation of UART wakeup feature.

- **uart_num** - UART number, the max port number is (UART_NUM_MAX -1).
- **out_wakeup_threshold** - [out] output, set to the current value of wakeup threshold for the given UART.

**返回**
- ESP_OK on success
**Chapter 2. API 参考**

- ESP_ERR_INVALID_ARG if out_wakeup_threshold is NULL

```c
esp_err_t uart_wait_tx_idle_polling(uart_port_t uart_num)
```

Wait until UART tx memory empty and the last char send ok (polling mode).

- ESP_OK on success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver not installed

**参数**
- `uart_num` - UART number

```c
esp_err_t uart_set_loop_back(uart_port_t uart_num, bool loop_back_en)
```

Configure TX signal loop back to RX module, just for the test usage.

- ESP_OK on success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver not installed

**参数**
- `uart_num` - UART number
- `loop_back_en` - Set true to enable the loop back function, else set it false.

```c
void uart_set_always_rx_timeout(uart_port_t uart_num, bool always_rx_timeout_en)
```

Configure behavior of UART RX timeout interrupt.

When `always_rx_timeout` is true, timeout interrupt is triggered even if FIFO is full. This function can cause extra timeout interrupts triggered only to send the timeout event. Call this function only if you want to ensure timeout interrupt will always happen after a byte stream.

**参数**
- `uart_num` - UART number
- `always_rx_timeout_en` - Set to false enable the default behavior of timeout interrupt, set it to true to always trigger timeout interrupt.

**Structures**

```c
struct uart_intr_config_t
```

UART interrupt configuration parameters for `uart_intr_config` function.

**Public Members**

```c
uint32_t intr_enable_mask
```

UART interrupt enable mask, choose from `UART_XXXX_INT_ENA_M` under `UART_INT_ENA_REG(i)`, connect with bit-or operator

```c
uint8_t rx_timeout_thresh
```

UART timeout interrupt threshold (unit: time of sending one byte)
Chapter 2. API

```c
uint8_t txfifo_empty_intr_thresh
UART TX empty interrupt threshold.

uint8_t rxfifo_full_thresh
UART RX full interrupt threshold.
```

```c
struct uart_event_t
Event structure used in UART event queue.
```

**Public Members**

```c
uart_event_type_t type
UART event type
```

```c
size_t size
UART data size for UART_DATA event
```

```c
bool timeout_flag
UART data read timeout flag for UART_DATA event (no new data received during configured RX TOUT) If the event is caused by FIFO-full interrupt, then there will be no event with the timeout flag before the next byte coming.
```

**Macros**

```c
UART_NUM_0
UART port 0
```

```c
UART_NUM_1
UART port 1
```

```c
UART_NUM_2
UART port 2
```

```c
UART_NUM_MAX
UART port max
```

```c
UART_PIN_NO_CHANGE
```

**Type Definitions**

```c
typedef intr_handle_t uart_isr_handle_t
```

Submit Document Feedback
Enumerations

enum uart_event_type_t

UART event types used in the ring buffer.

Values:

enumerator UART_DATA

UART data event

enumerator UART_BREAK

UART break event

enumerator UART_BUFFER_FULL

UART RX buffer full event

enumerator UART_FIFO_OVF

UART FIFO overflow event

enumerator UART_FRAME_ERR

UART RX frame error event

enumerator UART_PARITY_ERR

UART RX parity event

enumerator UART_DATA_BREAK

UART TX data and break event

enumerator UART_PATTERN_DET

UART pattern detected

enumerator UART_EVENT_MAX

UART event max index

Header File

- components/hal/include/hal/uart_types.h

Structures

struct uart_at_cmd_t

UART AT cmd char configuration parameters Note that this function may different on different chip. Please refer to the TRM at configuration.

Public Members

uint8_t cmd_char

UART AT cmd char
uint8_t char_num  
AT cmd char repeat number

uint32_t gap_tout  
gap time (in baud-rate) between AT cmd char

uint32_t pre_idle  
the idle time (in baud-rate) between the non AT char and first AT char

uint32_t post_idle  
the idle time (in baud-rate) between the last AT char and the none AT char

struct uart_sw_flowctrl_t  
UART software flow control configuration parameters.

**Public Members**

uint8_t xon_char  
Xon flow control char

uint8_t xoff_char  
Xoff flow control char

uint8_t xon_thrd  
If the software flow control is enabled and the data amount in rxfifo is less than xon_thrd, an xon_char will be sent

uint8_t xoff_thrd  
If the software flow control is enabled and the data amount in rxfifo is more than xoff_thrd, an xoff_char will be sent

struct uart_config_t  
UART configuration parameters for uart_param_config function.

**Public Members**

int baud_rate  
UART baud rate

_uart_word_length_t data_bits  
UART byte size

_uart_parity_t parity  
UART parity mode

_uart_stop_bits_t stop_bits  
UART stop bits
**Chapter 2. API**

```c
uart_hw_flowcontrol_t flow_ctrl
    UART HW flow control mode (cts/rts)

uint8_t rx_flow_ctrl_thresh
    UART HW RTS threshold

uart_sclk_t source_clk
    UART source clock selection
```

### Type Definitions

typedef int **uart_port_t**
    UART port number, can be UART_NUM_0 ~ (UART_NUM_MAX -1).

typedef **soc_periph_uart_clk_src_legacy_t** uart_sclk_t
    UART source clock.

### Enumerations

enum **uart_mode_t**
    UART mode selection.
    **Values:**

    enumerator **UART_MODE_UART**
        mode: regular UART mode

    enumerator **UART_MODE_RS485_HALF_DUPLEX**
        mode: half duplex RS485 UART mode control by RTS pin

    enumerator **UART_MODE_IRDA**
        mode: IRDA UART mode

    enumerator **UART_MODE_RS485_COLLISION_DETECT**
        mode: RS485 collision detection UART mode (used for test purposes)

    enumerator **UART_MODE_RS485_APP_CTRL**
        mode: application control RS485 UART mode (used for test purposes)

enum **uart_word_length_t**
    UART word length constants.
    **Values:**

    enumerator **UART_DATA_5_BITS**
        word length: 5bits

    enumerator **UART_DATA_6_BITS**
        word length: 6bits
enumerator UART_DATA_7_BITS
    word length: 7bits

enumerator UART_DATA_8_BITS
    word length: 8bits

enumerator UART_DATA_BITS_MAX

enum uart_stop_bits_t
    UART stop bits number.
    Values:
    
enumerator UART_STOP_BITS_1
    stop bit: 1bit

enumerator UART_STOP_BITS_1_5
    stop bit: 1.5bits

enumerator UART_STOP_BITS_2
    stop bit: 2bits

enumerator UART_STOP_BITS_MAX

enum uart_parity_t
    UART parity constants.
    Values:
    
enumerator UART_PARITY_DISABLE
    Disable UART parity

enumerator UART_PARITY_EVEN
    Enable UART even parity

enumerator UART_PARITY_ODD
    Enable UART odd parity

enum uart_hw_flowcontrol_t
    UART hardware flow control modes.
    Values:
    
enumerator UART_HW_FLOWCTRL_DISABLE
    disable hardware flow control

enumerator UART_HW_FLOWCTRL_RTS
    enable RX hardware flow control (rts)

enumerator UART_HW_FLOWCTRL_CTS
    enable TX hardware flow control (cts)
enumerator UART_HW_FLOWCTRL_CTS_RTS
    enable hardware flow control

enumerator UART_HW_FLOWCTRL_MAX

definitions

enum uart_signal_inv_t
    UART signal bit map.
    Values:

enumerator UART_SIGNAL_INV_DISABLE
    Disable UART signal inverse

enumerator UART_SIGNAL_IRDA_TX_INV
    inverse the UART irda_tx signal

enumerator UART_SIGNAL_IRDA_RX_INV
    inverse the UART irda_rx signal

enumerator UART_SIGNAL_RXD_INV
    inverse the UART rxd signal

enumerator UART_SIGNAL_CTS_INV
    inverse the UART cts signal

enumerator UART_SIGNAL_DSR_INV
    inverse the UART dsr signal

enumerator UART_SIGNAL_TXD_INV
    inverse the UART txd signal

enumerator UART_SIGNAL_RTS_INV
    inverse the UART rts signal

enumerator UART_SIGNAL_DTR_INV
    inverse the UART dtr signal

GPIOLookupMacros The UART peripherals have dedicated IO_MUX pins to which they are connected directly. However, signals can also be routed to other pins using the less direct GPIO matrix. To use direct routes, you need to know which pin is a dedicated IO_MUX pin for a UART channel. GPIOLookupMacros simplify the process of finding and assigning IO_MUX pins. You choose a macro based on either the IO_MUX pin number, or a required UART channel name, and the macro will return the matching counterpart for you. See some examples below.

备注：These macros are useful if you need very high UART baud rates (over 40 MHz), which means you will have to use IO_MUX pins only. In other cases, these macros can be ignored, and you can use the GPIO Matrix as it allows you to configure any GPIO pin for any UART function.

1. UART_NUM_2_TXD_DIRECT_GPIO_NUM returns the IO_MUX pin number of UART channel 2 TXD pin (pin 17)
2. **UART_GPIO19_DIRECT_CHANNEL** returns the UART number of GPIO 19 when connected to the UART peripheral via IO_MUX (this is UART_NUM_0).

3. **UART_CTS_GPIO19_DIRECT_CHANNEL** returns the UART number of GPIO 19 when used as the UART CTS pin via IO_MUX (this is UART_NUM_0). Similar to the above macro but specifies the pin function which is also part of the IO_MUX assignment.

### Header File

- `components/soc/esp32/include/soc/uart_channel.h`

### Macros

- `UART_GPIO1_DIRECT_CHANNEL`
- `UART_NUM_0_TXD_DIRECT_GPIO_NUM`
- `UART_GPIO3_DIRECT_CHANNEL`
- `UART_NUM_0_RXD_DIRECT_GPIO_NUM`
- `UART_GPIO19_DIRECT_CHANNEL`
- `UART_NUM_0_CTS_DIRECT_GPIO_NUM`
- `UART_GPIO22_DIRECT_CHANNEL`
- `UART_NUM_0_RTS_DIRECT_GPIO_NUM`
- `UART_TXD_GPIO1_DIRECT_CHANNEL`
- `UART_RXD_GPIO3_DIRECT_CHANNEL`
- `UART_CTS_GPIO19_DIRECT_CHANNEL`
- `UART_RTS_GPIO22_DIRECT_CHANNEL`
- `UART_GPIO10_DIRECT_CHANNEL`
- `UART_NUM_1_TXD_DIRECT_GPIO_NUM`
- `UART_GPIO9_DIRECT_CHANNEL`
- `UART_NUM_1_RXD_DIRECT_GPIO_NUM`
- `UART_GPIO6_DIRECT_CHANNEL`
- `UART_NUM_1_CTS_DIRECT_GPIO_NUM`
- `UART_GPIO10_DIRECT_CHANNEL`
UART_GPIO11_DIRECT_CHANNEL
UART_NUM_1_RTS_DIRECT_GPIO_NUM
UART_TXD_GPIO10_DIRECT_CHANNEL
UART_RXD_GPIO9_DIRECT_CHANNEL
UART_CTS_GPIO6_DIRECT_CHANNEL
UART_RTS_GPIO11_DIRECT_CHANNEL
UART_GPIO17_DIRECT_CHANNEL
UART_NUM_2_TXD_DIRECT_GPIO_NUM
UART_GPIO16_DIRECT_CHANNEL
UART_NUM_2_RXD_DIRECT_GPIO_NUM
UART_GPIO8_DIRECT_CHANNEL
UART_NUM_2_CTS_DIRECT_GPIO_NUM
UART_GPIO7_DIRECT_CHANNEL
UART_NUM_2_RTS_DIRECT_GPIO_NUM
UART_TXD_GPIO17_DIRECT_CHANNEL
UART_RXD_GPIO16_DIRECT_CHANNEL
UART_CTS_GPIO8_DIRECT_CHANNEL
UART_RTS_GPIO7_DIRECT_CHANNEL

2.7 Project Configuration

2.7.1 Introduction

ESP-IDF uses the esp-idf-kconfig package based on kconfiglib which is a Python-based extension to the Kconfig system. Kconfig provides a compile-time project configuration mechanism and is based around options of several types: integer, string, boolean. Kconfig files specify dependencies between options, default values of the options, the way the options are grouped together, etc.

For the complete list of available features please see Kconfig and kconfiglib extentions.
2.7.2 Project Configuration Menu

Application developers can open a terminal-based project configuration menu with the `idf.py menuconfig` build target.

After being updated, this configuration is saved inside `sdkconfig` file in the project root directory. Based on `sdkconfig`, application build targets will generate `sdkconfig.h` file in the build directory, and will make `sdkconfig` options available to the project build system and source files.

2.7.3 Using `sdkconfig.defaults`

In some cases, such as when `sdkconfig` file is under revision control, the fact that `sdkconfig` file gets changed by the build system may be inconvenient. The build system offers a way to avoid this, in the form of `sdkconfig.defaults` file. This file is never touched by the build system, and can be created manually or automatically. It can contain all the options which matter for the given application and are different from the default ones. The format is the same as that of the `sdkconfig` file. `sdkconfig.defaults` can be created manually when one remembers all the changed configurations. Otherwise, the file can be generated automatically by running the `idf.py save-defconfig` command.

Once `sdkconfig.defaults` is created, `sdkconfig` can be deleted and added to the ignore list of the revision control system (e.g. `.gitignore` file for `git`). Project build targets will automatically create `sdkconfig` file, populated with the settings from `sdkconfig.defaults` file, and the rest of the settings will be set to their default values. Note that the build process will not override settings that are already in `sdkconfig` by ones from `sdkconfig.defaults`. For more information, see 自定义 `sdkconfig` 的默认值.

2.7.4 Kconfig Formatting Rules

The following attributes of `Kconfig` files are standardized:

- Within any menu, option names should have a consistent prefix. The prefix length is currently set to at least 3 characters.
- The indentation style is 4 characters created by spaces. All sub-items belonging to a parent item are indented by one level deeper. For example, `menu` is indented by 0 characters, the `config` inside of the menu by 4 characters, the help of the `config` by 8 characters and the text of the `help` by 12 characters.
- No trailing spaces are allowed at the end of the lines.
- The maximum length of options is set to 40 characters.
- The maximum length of lines is set to 120 characters.

Format checker

`tools/check_kconfigs.py` is provided for checking the `Kconfig` formatting rules. The checker checks all `Kconfig` and `Kconfig.projbuild` files in the ESP-IDF directory and generates a new file with suffix `.new` with some recommendations how to fix issues (if there are any). Please note that the checker cannot correct all rules and the responsibility of the developer is to check and make final corrections in order to pass the tests. For example, indentations will be corrected if there isn’t some misleading previous formatting but it cannot come up with a common prefix for options inside a menu.

2.7.5 Backward Compatibility of Kconfig Options

The standard `Kconfig` tools ignore unknown options in `sdkconfig`. So if a developer has custom settings for options which are renamed in newer ESP-IDF releases then the given setting for the option would be silently ignored. Therefore, several features have been adopted to avoid this:

1. `kconfiggen` is used by the tool chain to pre-process `sdkconfig` files before anything else, for example `menuconfig` would read them. As the consequence, the settings for old options will be kept and not ignored.
2. `kconfgen` recursively finds all `sdkconfig.rename` files in ESP-IDF directory which contain old and new Kconfig option names. Old options are replaced by new ones in the `sdkconfig` file. Renames that should only appear for a single target can be placed in a target specific rename file: `sdkconfig.rename.TARGET`, where `TARGET` is the target name, e.g. `sdkconfig.rename.esp32s2`.

3. `kconfgen` post-processes `sdkconfig` files and generates all build outputs (`sdkconfig.h`, `sdkconfig.cmake`, `auto.conf`) by adding a list of compatibility statements, i.e. value of the old option is set the value of the new option (after modification). This is done in order to not break customer codes where old option might still be used.

4. **Deprecated options and their replacements** are automatically generated by `kconfgen`.

### 2.7.6 Configuration Options Reference

Subsequent sections contain the list of available ESP-IDF options, automatically generated from Kconfig files. Note that depending on the options selected, some options listed here may not be visible by default in the interface of `menuconfig`.

By convention, all option names are upper case with underscores. When Kconfig generates `sdkconfig` and `sdkconfig.h` files, option names are prefixed with `CONFIG_`. So if an option `ENABLE_FOO` is defined in a Kconfig file and selected in `menuconfig`, then `sdkconfig` and `sdkconfig.h` files will have `CONFIG_ENABLE_FOO` defined. In this reference, option names are also prefixed with `CONFIG_`, same as in the source code.

**Build type**

Contains:

- `CONFIG_APP_COMPATIBLE_PRE_V3_1_BOOTLOADERS`
- `CONFIG_APP_COMPATIBLE_PRE_V2_1_BOOTLOADERS`
- `CONFIG_APP_BUILD_TYPE`
- `CONFIG_APP_REPRODUCIBLE_BUILD`
- `CONFIG_APP_NO_BLOBS`

**CONFIG_APP_BUILD_TYPE**

Application build type

*Found in: Build type*

Select the way the application is built.

By default, the application is built as a binary file in a format compatible with the ESP-IDF bootloader. In addition to this application, 2nd stage bootloader is also built. Application and bootloader binaries can be written into flash and loaded/executed from there.

Another option, useful for only very small and limited applications, is to only link the .elf file of the application, such that it can be loaded directly into RAM over JTAG. Note that since IRAM and DRAM sizes are very limited, it is not possible to build any complex application this way. However for kinds of testing and debugging, this option may provide faster iterations, since the application does not need to be written into flash. Note that at the moment, ESP-IDF does not contain all the startup code required to initialize the CPUs and ROM memory (data/bss). Therefore it is necessary to execute a bit of ROM code prior to executing the application. A gdbinit file may look as follows (for ESP32):

```bash
# Connect to a running instance of OpenOCD target remote :3333 # Reset and halt the target
mon reset halt # Run to a specific point in ROM code, # where most of initialization is complete. thb *0x40007d54 c # Load the application into RAM load # Run till app_main tb
app_main c
```

Execute this gdbinit file as follows:

```bash
xtensa-esp32-elf-gdb build/app-name.elf -x gdbinit
```
Example gdbinit files for other targets can be found in tools/test_apps/system/gdb_loadable_elf/

Recommended sdkconfig.defaults for building loadable ELF files is as follows. CONFIG_APP_BUILD_TYPE_ELF_RAM is required, other options help reduce application memory footprint.

```cpp
CONFIG_APP_BUILD_TYPE_ELF_RAM=y
CONFIG_VFS_SUPPORT_TERMIOS=y
CONFIG_NEWLIB_NANO_FORMAT=y
CONFIG_ESP_SYSTEM_PANIC_PRINT_HALT=y
CONFIG_ESP_DEBUG_STUBS_ENABLE=y
CONFIG_ESP_ERR_TO_NAME_LOOKUP=y
```

Available options:
- Default (binary application + 2nd stage bootloader) (APP_BUILD_TYPE_APP_2NDBOOT)
- ELF file, loadable into RAM (EXPERIMENTAL) (APP_BUILD_TYPE_ELF_RAM)

**CONFIG_APP_REPRODUCIBLE_BUILD**

Enable reproducible build

*Found in: Build type*

If enabled, all date, time, and path information would be eliminated. A .gdbinit file would be create automatically. (or will be append if you have one already)

*Default value:*
- No (disabled)

**CONFIG_APP_NO_BLOBS**

No Binary Blobs

*Found in: Build type*

If enabled, this disables the linking of binary libraries in the application build. Note that after enabling this Wi-Fi/Bluetooth will not work.

*Default value:*
- No (disabled)

**CONFIG_APP_COMPATIBLE_PRE_V2_1_BOOTLOADERs**

App compatible with bootloader before ESP-IDF v2.1

*Found in: Build type*

Bootloaders before ESP-IDF v2.1 did less initialisation of the system clock. This setting needs to be enabled to build an app which can be booted by these older bootloaders.

- If this setting is enabled, the app can be booted by any bootloader from IDF v1.0 up to the current version.
- If this setting is disabled, the app can only be booted by bootloaders from IDF v2.1 or newer.

Enabling this setting adds approximately 1KB to the app’s IRAM usage.

*Default value:*
- No (disabled)

**CONFIG_APP_COMPATIBLE_PRE_V3_1_BOOTLOADERs**

App compatible with bootloader and partition table before ESP-IDF v3.1

*Found in: Build type*

Partition tables before ESP-IDF V3.1 do not contain an MD5 checksum field, and the bootloader before ESP-IDF v3.1 cannot read a partition table that contains an MD5 checksum field.
Enable this option only if your app needs to boot on a bootloader and/or partition table that was generated from a version *before* ESP-IDF v3.1.

If this option and Flash Encryption are enabled at the same time, and any data partitions in the partition table are marked Encrypted, then the partition encrypted flag should be manually verified in the app before accessing the partition (see CVE-2021-27926).

**Default value:**
- No (disabled)

**Bootloader config**

Contains:

- `CONFIG_BOOTLOADER_LOG_LEVEL`
- `CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION`
- `CONFIG_BOOTLOADER_SPI_WP_PIN`
- `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE`
- `CONFIG_BOOTLOADER_REGION_PROTECTION_ENABLE`
- `CONFIG_BOOTLOADER_FLASH_XMC_SUPPORT`
- `CONFIG_BOOTLOADER_APP_TEST`
- `CONFIG_BOOTLOADER_FACTORY_RESET`
- `CONFIG_BOOTLOADER_HOLD_TIME_GPIO`
- `CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC`
- `CONFIG_BOOTLOADER_SKIP_VALIDATE_ALWAYS`
- `CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON`
- `CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP`
- `CONFIG_BOOTLOADER_SPI_CUSTOM_WP_PIN`
- `CONFIG_BOOTLOADER_WDT_ENABLE`
- `CONFIG_BOOTLOADER_VDSDIO_BOOST`

**CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION**

Bootloader optimization Level

*Found in: Bootloader config*

This option sets compiler optimization level (gcc `O` argument) for the bootloader.

- The default “Size” setting will add the `-Os` flag to CFLAGS.
- The “Debug” setting will add the `-Og` flag to CFLAGS.
- The “Performance” setting will add the `-O2` flag to CFLAGS.
- The “None” setting will add the `-O0` flag to CFLAGS.

Note that custom optimization levels may be unsupported.

**Available options:**

- Size (-Os) (BOOTLOADER_COMPILER_OPTIMIZATION_SIZE)
- Debug (-Og) (BOOTLOADER_COMPILER_OPTIMIZATION_DEBUG)
- Optimize for performance (-O2) (BOOTLOADER_COMPILER_OPTIMIZATION_PERF)
- Debug without optimization (-O0) (BOOTLOADER_COMPILER_OPTIMIZATION_NONE)

**CONFIG_BOOTLOADER_LOG_LEVEL**

Bootloader log verbosity

*Found in: Bootloader config*

Specify how much output to see in bootloader logs.

**Available options:**

- No output (BOOTLOADER_LOG_LEVEL_NONE)
• Error (BOOTLOADER_LOG_LEVEL_ERROR)
• Warning (BOOTLOADER_LOG_LEVEL_WARN)
• Info (BOOTLOADER_LOG_LEVEL_INFO)
• Debug (BOOTLOADER_LOG_LEVEL_DEBUG)
• Verbose (BOOTLOADER_LOG_LEVEL_VERBOSE)

**CONFIG_BOOTLOADER_SPI_CUSTOM_WP_PIN**

Use custom SPI Flash WP Pin when flash pins set in eFuse (read help)

*Found in: Bootloader config*

This setting is only used if the SPI flash pins have been overridden by setting the eFuses SPI_PAD_CONFIG_xxx, and the SPI flash mode is QIO or QOUT.

When this is the case, the eFuse config only defines 3 of the 4 Quad I/O data pins. The WP pin (aka ESP32 pin “SD_DATA_3” or SPI flash pin “IO2”) is not specified in eFuse. The same pin is also used for external SPIRAM if it is enabled.

If this config item is set to N (default), the correct WP pin will be automatically used for any Espressif chip or module with integrated flash. If a custom setting is needed, set this config item to Y and specify the GPIO number connected to the WP.

*Default value:*

• No (disabled) if ESPTOOLPY_FLASHMODE_QIO || ESPTOOLPY_FLASHMODE_QOUT

**CONFIG_BOOTLOADER_SPI_WP_PIN**

Custom SPI Flash WP Pin

*Found in: Bootloader config*

The option “Use custom SPI Flash WP Pin” must be set or this value is ignored.

If burning a customized set of SPI flash pins in eFuse and using QIO or QOUT mode for flash, set this value to the GPIO number of the SPI flash WP pin.

*Range:*

• from 0 to 33 if ESPTOOLPY_FLASHMODE_QIO || ESPTOOLPY_FLASHMODE_QOUT

*Default value:*

• 7 if ESPTOOLPY_FLASHMODE_QIO || ESPTOOLPY_FLASHMODE_QOUT

**CONFIG_BOOTLOADER_VDDSDIO_BOOST**

VDDSDIO LDO voltage

*Found in: Bootloader config*

If this option is enabled, and VDDSDIO LDO is set to 1.8V (using eFuse or MT-DI bootstrapping pin), bootloader will change LDO settings to output 1.9V instead. This helps prevent flash chip from browning out during flash programming operations.

This option has no effect if VDDSDIO is set to 3.3V, or if the internal VDDSDIO regulator is disabled via eFuse.

*Available options:*

• 1.8V (BOOTLOADER_VDDSDIO_BOOST_1_8V)
• 1.9V (BOOTLOADER_VDDSDIO_BOOST_1_9V)
**CONFIG_BOOTLOADER_FACTORY_RESET**

GPIO triggers factory reset

*Found in: Bootloader config*

Allows to reset the device to factory settings: - clear one or more data partitions; - boot from “factory” partition. The factory reset will occur if there is a GPIO input held at the configured level while device starts up. See settings below.

**Default value:**
- No (disabled)

**CONFIG_BOOTLOADER_NUM_PIN_FACTORY_RESET**

Number of the GPIO input for factory reset

*Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET*

The selected GPIO will be configured as an input with internal pull-up enabled (note that on some SoCs, not all pins have an internal pull-up, consult the hardware datasheet for details.) To trigger a factory reset, this GPIO must be held high or low (as configured) on startup.

**Range:**
- from 0 to 39 if `CONFIG_BOOTLOADER_FACTORY_RESET`

**Default value:**
- 4 if `CONFIG_BOOTLOADER_FACTORY_RESET`

**CONFIG_BOOTLOADER_FACTORY_RESET_PIN_LEVEL**

Factory reset GPIO level

*Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET*

Pin level for factory reset, can be triggered on low or high.

**Available options:**
- Reset on GPIO low (BOOTLOADER_FACTORY_RESET_PIN_LOW)
- Reset on GPIO high (BOOTLOADER_FACTORY_RESET_PIN_HIGH)

**CONFIG_BOOTLOADER_OTA_DATA_ERASE**

Clear OTA data on factory reset (select factory partition)

*Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET*

The device will boot from “factory” partition (or OTA slot 0 if no factory partition is present) after a factory reset.

**CONFIG_BOOTLOADER_DATA_FACTORY_RESET**

Comma-separated names of partitions to clear on factory reset

*Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET*

Allows customers to select which data partitions will be erased while factory reset.

Specify the names of partitions as a comma-delimited with optional spaces for readability. (Like this: “nvs, phy_init, …”) Make sure that the name specified in the partition table and here are the same. Partitions of type “app” cannot be specified here.

**Default value:**
- “nvs” if `CONFIG_BOOTLOADER_FACTORY_RESET`
**CONFIG_BOOTLOADER_APP_TEST**

GPIO triggers boot from test app partition

*Found in: Bootloader config*

Allows to run the test app from “TEST” partition. A boot from “test” partition will occur if there is a GPIO input pulled low while device starts up. See settings below.

**Default value:**
- No (disabled) if `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK`

**CONFIG_BOOTLOADER_NUM_PIN_APP_TEST**

Number of the GPIO input to boot TEST partition

*Found in: Bootloader config > CONFIG_BOOTLOADER_APP_TEST*

The selected GPIO will be configured as an input with internal pull-up enabled. To trigger a test app, this GPIO must be pulled low on reset. After the GPIO input is deactivated and the device reboots, the old application will boot. (factory or OTA[x]). Note that GPIO34-39 do not have an internal pullup and an external one must be provided.

**Range:**
- from 0 to 39 if `CONFIG_BOOTLOADER_APP_TEST`

**Default value:**
- 18 if `CONFIG_BOOTLOADER_APP_TEST`

**CONFIG_BOOTLOADER_APP_TEST_PIN_LEVEL**

App test GPIO level

*Found in: Bootloader config > CONFIG_BOOTLOADER_APP_TEST*

Pin level for app test, can be triggered on low or high.

**Available options:**
- Enter test app on GPIO low (BOOTLOADER_APP_TEST_PIN_LOW)
- Enter test app on GPIO high (BOOTLOADER_APP_TEST_PIN_HIGH)

**CONFIG_BOOTLOADER_HOLD_TIME_GPIO**

Hold time of GPIO for reset/test mode (seconds)

*Found in: Bootloader config*

The GPIO must be held low continuously for this period of time after reset before a factory reset or test partition boot (as applicable) is performed.

**Default value:**
- 5 if `CONFIG_BOOTLOADER_FACTORY_RESET || CONFIG_BOOTLOADER_APP_TEST`

**CONFIG_BOOTLOADER_REGION_PROTECTION_ENABLE**

Enable protection for unmapped memory regions

*Found in: Bootloader config*

Protects the unmapped memory regions of the entire address space from unintended accesses. This will ensure that an exception will be triggered whenever the CPU performs a memory operation on unmapped regions of the address space.

**Default value:**
- Yes (enabled)
**CONFIG_BOOTLOADER_WDT_ENABLE**

Use RTC watchdog in start code

*Found in: Bootloader config*

Tracks the execution time of startup code. If the execution time is exceeded, the RTC_WDT will restart system. It is also useful to prevent a lock up in start code caused by an unstable power source. NOTE: Tracks the execution time starts from the bootloader code - re-set timeout, while selecting the source for slow_clk - and ends calling app_main. Re-set timeout is needed due to WDT uses a SLOW_CLK clock source. After changing a frequency slow_clk a time of WDT needs to re-set for new frequency. slow_clk depends on RTC_CLK_SRC (INTERNAL_RC or EXTERNALCRYSTAL).

*Default value:*
- Yes (enabled)

**CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE**

Allows RTC watchdog disable in user code

*Found in: Bootloader config > CONFIG_BOOTLOADER_WDT_ENABLE*

If this option is set, the ESP-IDF app must explicitly reset, feed, or disable the rtc_wdt in the app’s own code. If this option is not set (default), then rtc_wdt will be disabled by ESP-IDF before calling the app_main() function.

Use function rtc_wdt_feed() for resetting counter of rtc_wdt. Use function rtc_wdt_disable() for disabling rtc_wdt.

*Default value:*
- No (disabled)

**CONFIG_BOOTLOADER_WDT_TIME_MS**

Timeout for RTC watchdog (ms)

*Found in: Bootloader config > CONFIG_BOOTLOADER_WDT_ENABLE*

Verify that this parameter is correct and more then the execution time. Pay attention to options such as reset to factory, trigger test partition and encryption on boot - these options can increase the execution time. Note: RTC_WDT will reset while encryption operations will be performed.

*Range:*
- from 0 to 120000

*Default value:*
- 9000

**CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE**

Enable app rollback support

*Found in: Bootloader config*

After updating the app, the bootloader runs a new app with the “ESP_OTA_IMG_PENDING_VERIFY” state set. This state prevents the re-run of this app. After the first boot of the new app in the user code, the function should be called to confirm the operability of the app or vice versa about its non-operability. If the app is working, then it is marked as valid. Otherwise, it is marked as not valid and rolls back to the previous working app. A reboot is performed, and the app is booted before the software update. Note: If during the first boot a new app the power goes out or the WDT works, then roll back will happen. Rollback is possible only between the apps with the same security versions.

*Default value:*
- No (disabled)
**CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK**

Enable app anti-rollback support

*Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*

This option prevents rollback to previous firmware/application image with lower security version.

**Default value:**
- No (disabled) if `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE`

**CONFIG_BOOTLOADER_APP_SECURE_VERSION**

eFuse secure version of app

*Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

The secure version is the sequence number stored in the header of each firmware. The security version is set in the bootloader, version is recorded in the eFuse field as the number of set ones. The allocated number of bits in the efuse field for storing the security version is limited (see `BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD` option).

Bootloader: When bootloader selects an app to boot, an app is selected that has a security version greater or equal that recorded in eFuse field. The app is booted with a higher (or equal) secure version.

The security version is worth increasing if in previous versions there is a significant vulnerability and their use is not acceptable.

Your partition table should have a scheme with ota_0 + ota_1 (without factory).

**Default value:**
- 0 if `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK`

**CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD**

Size of the eFuse secure version field

*Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

The size of the eFuse secure version field. Its length is limited to 32 bits for ESP32 and 16 bits for ESP32-S2. This determines how many times the security version can be increased.

**Range:**
- from 1 to 32 if `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK`
- from 1 to 16 if `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK`

**Default value:**
- 32 if `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK`
- 16 if `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK`

**CONFIG_BOOTLOADER_EFUSE_SECURE_VERSION_EMULATE**

Emulate operations with efuse secure version (only test)

*Found in: Bootloader config > CONFIG BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*

This option allows to emulate read/write operations with all eFuses and efuse secure version. It allows to test anti-rollback implementation without permanent write eFuse bits. There should be an entry in partition table with following details: `emul_efuse, data, efuse, , 0x2000`.

This option enables: EFUSE_VIRTUAL and EFUSE_VIRTUAL_KEEP_IN_FLASH.

**Default value:**
- No (disabled) if `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK`
CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP

Skip image validation when exiting deep sleep

Found in: Bootloader config

This option disables the normal validation of an image coming out of deep sleep (checksums, SHA256, and signature). This is a trade-off between wakeup performance from deep sleep, and image integrity checks.

Only enable this if you know what you are doing. It should not be used in conjunction with using deep_sleep() entry and changing the active OTA partition as this would skip the validation upon first load of the new OTA partition.

It is possible to enable this option with Secure Boot if “allow insecure options” is enabled, however it’s strongly recommended to NOT enable it as it may allow a Secure Boot bypass.

Default value:
- No (disabled) if (CONFIG_SECURE_BOOT && CONFIG_SECURE_BOOT_INSECURE) || CONFIG_SECURE_BOOT

CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON

Skip image validation from power on reset (READ HELP FIRST)

Found in: Bootloader config

Some applications need to boot very quickly from power on. By default, the entire app binary is read from flash and verified which takes up a significant portion of the boot time.

Enabling this option will skip validation of the app when the SoC boots from power on. Note that in this case it’s not possible for the bootloader to detect if an app image is corrupted in the flash, therefore it’s not possible to safely fall back to a different app partition. Flash corruption of this kind is unlikely but can happen if there is a serious firmware bug or physical damage.

Following other reset types, the bootloader will still validate the app image. This increases the chances that flash corruption resulting in a crash can be detected following soft reset, and the bootloader will fall back to a valid app image. To increase the chances of successfully recovering from a flash corruption event, keep the option BOOTLOADER_WDT_ENABLE enabled and consider also enabling BOOTLOADER_WDT_DISABLE_IN_USER_CODE - then manually disable the RTC Watchdog once the app is running. In addition, enable both the Task and Interrupt watchdog timers with reset options set.

Default value:
- No (disabled)

CONFIG_BOOTLOADER_SKIP_VALIDATE_ALWAYS

Skip image validation always (READ HELP FIRST)

Found in: Bootloader config

Selecting this option prevents the bootloader from ever validating the app image before booting it. Any flash corruption of the selected app partition will make the entire SoC unbootable.

Although flash corruption is a very rare case, it is not recommended to select this option. Consider selecting “Skip image validation from power on reset” instead. However, if boot time is the only important factor then it can be enabled.

Default value:
- No (disabled)
**CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC**

Reserve RTC FAST memory for custom purposes

*Found in: Bootloader config*

This option allows the customer to place data in the RTC FAST memory, this area remains valid when rebooted, except for power loss. This memory is located at a fixed address and is available for both the bootloader and the application. (The application and bootloader must be compiled with the same option). The RTC FAST memory has access only through PRO_CPU.

**Default value:**
- No (disabled)

**CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC_SIZE**

Size in bytes for custom purposes

*Found in: Bootloader config > CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC*

This option reserves in RTC FAST memory the area for custom purposes. If you want to create your own bootloader and save more information in this area of memory, you can increase it. It must be a multiple of 4 bytes. This area (rtc_retain_mem_t) is reserved and has access from the bootloader and an application.

**Default value:**
- 0 if CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC

**CONFIG_BOOTLOADER_FLASH_XMC_SUPPORT**

Enable the support for flash chips of XMC (READ HELP FIRST)

*Found in: Bootloader config*

Perform the startup flow recommended by XMC. Please consult XMC for the details of this flow. XMC chips will be forbidden to be used, when this option is disabled.

**Default value:**
- Yes (enabled)

**Security features**

Contains:

- CONFIG_SECURE_BOOT_INSECURE
- CONFIG_SECURE_SIGNED_APPS_SCHEME
- CONFIG_SECURE_SIGNED_ON_BOOT_NO_SECURE_BOOT
- CONFIG_SECURE_FLASH_CHECK_ENC_EN_IN_APP
- CONFIG_SECURE_BOOT_ECDSA_KEY_LEN_SIZE
- CONFIG_SECURE_BOOT_ENABLE_AGGRESSIVE_KEY_REVOKE
- CONFIG_SECURE_FLASH_ENC_ENABLED
- CONFIG_SECURE_BOOT
- CONFIG_SECURE_BOOTLOADER_KEY_ENCODING
- Potentially insecure options
- CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT
- CONFIG_SECURE_BOOT_VERIFICATION_KEY
- CONFIG_SECURE_BOOTLOADER_MODE
- CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES
- CONFIG_SECURE_UART_ROM_DL_MODE
- CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT
CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT

Require signed app images

*Found in: Security features*

Require apps to be signed to verify their integrity.

This option uses the same app signature scheme as hardware secure boot, but unlike hardware secure boot it does not prevent the bootloader from being physically updated. This means that the device can be secured against remote network access, but not physical access. Compared to using hardware Secure Boot this option is much simpler to implement.

CONFIG_SECURE_SIGNED_APPS_SCHEME

App Signing Scheme

*Found in: Security features*

Select the Secure App signing scheme. Depends on the Chip Revision. There are two secure boot versions:

1. **Secure boot V1**
   - Legacy custom secure boot scheme. Supported in ESP32 SoC.

2. **Secure boot V2**
   - RSA based secure boot scheme. Supported in ESP32-ECO3 (ESP32 Chip Revision 3 onwards), ESP32-S2, ESP32-C3, ESP32-S3 SoCs.
   - ECDSA based secure boot scheme. Supported in ESP32-C2 SoC.

*Available options:*

- **ECDSA (SECURE_SIGNED_APPS_ECDSA_SCHEME)**
  - Embeds the ECDSA public key in the bootloader and signs the application with an ECDSA key. Refer to the documentation before enabling.

- **RSA (SECURE_SIGNED_APPS_RSA_SCHEME)**
  - Appends the RSA-3072 based Signature block to the application. Refer to <Secure Boot Version 2 documentation link> before enabling.

- **ECDSA (V2) (SECURE_SIGNED_APPS_ECDSA_V2_SCHEME)**
  - For Secure boot V2 (e.g., ESP32-C2 SoC), appends ECDSA based signature block to the application. Refer to documentation before enabling.

CONFIG_SECURE_BOOT_ECDSA_KEY_LEN_SIZE

ECDSA key size

*Found in: Security features*

Select the ECDSA key size. Two key sizes are supported

- 192 bit key using NISTP192 curve
- 256 bit key using NISTP256 curve (Recommended)

The advantage of using 256 bit key is the extra randomness which makes it difficult to be bruteforced compared to 192 bit key. At present, both key sizes are practically implausible to brute force.

*Available options:*

- Using ECC curve NISTP192 (SECURE_BOOT_ECDSA_KEY_LEN_192_BITS)
- Using ECC curve NISTP256 (Recommended) (SECURE_BOOT_ECDSA_KEY_LEN_256_BITS)

CONFIG_SECURE_SIGNED_ON_BOOT_NO_SECURE_BOOT

Bootloader verifies app signatures

*Found in: Security features*
If this option is set, the bootloader will be compiled with code to verify that an app is signed before booting it.

If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option doesn’t add significant security by itself so most users will want to leave it disabled.

**Default value:**
- No (disabled) if `CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT` && `SECURE_SIGNED_APPS_ECDSA_SCHEME`

### CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT

Verify app signature on update

*Found in: Security features*

If this option is set, any OTA updated apps will have the signature verified before being considered valid.

When enabled, the signature is automatically checked whenever the esp_ota_ops.h APIs are used for OTA updates, or esp_image_format.h APIs are used to verify apps.

If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option still adds significant security against network-based attackers by preventing spoofing of OTA updates.

**Default value:**
- Yes (enabled) if `CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT`

### CONFIG_SECURE_BOOT

Enable hardware Secure Boot in bootloader (READ DOCS FIRST)

*Found in: Security features*

Build a bootloader which enables Secure Boot on first boot.

Once enabled, Secure Boot will not boot a modified bootloader. The bootloader will only load a partition table or boot an app if the data has a verified digital signature. There are implications for reflashing updated apps once secure boot is enabled.

When enabling secure boot, JTAG and ROM BASIC Interpreter are permanently disabled by default.

**Default value:**
- No (disabled)

### CONFIG_SECURE_BOOT_VERSION

Select secure boot version

*Found in: Security features > CONFIG_SECURE_BOOT*

Select the Secure Boot Version. Depends on the Chip Revision. Secure Boot V2 is the new RSA / ECDSA based secure boot scheme.

- RSA based scheme is supported in ESP32 (Revision 3 onwards), ESP32-S2, ESP32-C3 (ECO3), ESP32-S3.
- ECDSA based scheme is supported in ESP32-C2 SoC.

Please note that, RSA or ECDSA secure boot is property of specific SoC based on its HW design, supported crypto accelerators, die-size, cost and similar parameters. Please note that RSA scheme has requirement for bigger key sizes but at the same time it is comparatively faster than ECDSA verification.

Secure Boot V1 is the AES based (custom) secure boot scheme supported in ESP32 SoC.

**Available options:**

Chapter 2. API

- Enable Secure Boot version 1 (SECURE_BOOT_V1_ENABLED)
  Build a bootloader which enables secure boot version 1 on first boot. Refer to the Secure Boot section of the ESP-IDF Programmer’s Guide for this version before enabling.
- Enable Secure Boot version 2 (SECURE_BOOT_V2_ENABLED)
  Build a bootloader which enables Secure Boot version 2 on first boot. Refer to Secure Boot V2 section of the ESP-IDF Programmer’s Guide for this version before enabling.

**CONFIG_SECURE_BOOTLOADER_MODE**

Secure bootloader mode

*Found in: Security features*

**Available options:**

- One-time flash (SECURE_BOOTLOADER_ONE_TIME_FLASH)
  On first boot, the bootloader will generate a key which is not readable externally or by software. A digest is generated from the bootloader image itself. This digest will be verified on each subsequent boot. Enabling this option means that the bootloader cannot be changed after the first time it is booted.
- Reflashable (SECURE_BOOTLOADER_REFLASHABLE)
  Generate a reusable secure bootloader key, derived (via SHA-256) from the secure boot signing key. This allows the secure bootloader to be re-flashed by anyone with access to the secure boot signing key. This option is less secure than one-time flash, because a leak of the digest key from one device allows reflashing of any device that uses it.

**CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES**

Sign binaries during build

*Found in: Security features*

Once secure boot or signed app requirement is enabled, app images are required to be signed.

If enabled (default), these binary files are signed as part of the build process. The file named as “Secure boot private signing key” will be used to sign the image.

If disabled, unsigned app/partition data will be built. They must be signed manually using espsecure.py. Version 1 to enable ECDSA Based Secure Boot and Version 2 to enable RSA based Secure Boot. (for example, on a remote signing server.)

**CONFIG_SECURE_BOOT_SIGNING_KEY**

Secure boot private signing key

*Found in: Security features > CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES*

Path to the key file used to sign app images.

Key file is an ECDSA private key (NIST256p curve) in PEM format for Secure Boot V1. Key file is an RSA private key in PEM format for Secure Boot V2.

Path is evaluated relative to the project directory.

You can generate a new signing key by running the following command: espsecure.py generate_signing_key secure_boot_signing_key.pem

See the Secure Boot section of the ESP-IDF Programmer’s Guide for this version for details.

**Default value:**

- “secure_boot_signing_key.pem” if CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES
CONFIG_SECURE_BOOT_VERIFICATION_KEY

Secure boot public signature verification key

*Found in: Security features*

Path to a public key file used to verify signed images. Secure Boot V1: This ECDSA public key is compiled into the bootloader and/or app, to verify app images. Secure Boot V2: This RSA public key is compiled into the signature block at the end of the bootloader/app.

Key file is in raw binary format, and can be extracted from a PEM formatted private key using the espsecure.py extract_public_key command.

Refer to the Secure Boot section of the ESP-IDF Programmer’s Guide for this version before enabling.

CONFIG_SECURE_BOOT_ENABLE_AGGRESSIVE_KEY_REVOKE

Enable Aggressive key revoke strategy

*Found in: Security features*

If this option is set, ROM bootloader will revoke the public key digest burned in efuse block if it fails to verify the signature of software bootloader with it. Revocation of keys does not happen when enabling secure boot. Once secure boot is enabled, key revocation checks will be done on subsequent boot-up, while verifying the software bootloader.

This feature provides a strong resistance against physical attacks on the device.

NOTE: Once a digest slot is revoked, it can never be used again to verify an image. This can lead to permanent bricking of the device, in case all keys are revoked because of signature verification failure.

*Default value:*

- No (disabled) if `CONFIG_SECURE_BOOT && SOC_SUPPORT_SECURE_BOOT_REVOKE_KEY`

CONFIG_SECURE_BOOTLOADER_KEY_ENCODING

Hardware Key Encoding

*Found in: Security features*

In reflashable secure bootloader mode, a hardware key is derived from the signing key (with SHA-256) and can be written to eFuse with espfuse.py.

Normally this is a 256-bit key, but if 3/4 Coding Scheme is used on the device then the eFuse key is truncated to 192 bits.

This configuration item doesn’t change any firmware code, it only changes the size of key binary which is generated at build time.

*Available options:*

- No encoding (256 bit key) (SECURE_BOOTLOADER_KEY_ENCODING_256BIT)
- 3/4 encoding (192 bit key) (SECURE_BOOTLOADER_KEY_ENCODING_192BIT)

CONFIG_SECURE_BOOT_INSECURE

Allow potentially insecure options

*Found in: Security features*

You can disable some of the default protections offered by secure boot, in order to enable testing or a custom combination of security features.

Only enable these options if you are very sure.

Refer to the Secure Boot section of the ESP-IDF Programmer’s Guide for this version before enabling.

*Default value:*
Chapter 2. API

• No (disabled) if `CONFIG_SECURE_BOOT`

**CONFIG_SECURE_FLASH_ENC_ENABLED**

Enable flash encryption on boot (READ DOCS FIRST)

*Found in: Security features*

If this option is set, flash contents will be encrypted by the bootloader on first boot.

Note: After first boot, the system will be permanently encrypted. Re-flashing an encrypted system is complicated and not always possible.

Read [*Flash Encryption*](https://github.com/espressif/esp-idf) before enabling.

**Default value:**

• No (disabled)

**CONFIG_SECURE_FLASH_ENCRYPTION_KEYSIZE**

Size of generated AES-XTS key

*Found in: Security features > CONFIG_SECURE_FLASH_ENC_ENABLED*

Size of generated AES-XTS key.

• AES-128 uses a 256-bit key (32 bytes) derived from 128 bits (16 bytes) burned in half Efuse key block. Internally, it calculates SHA256(128 bits)
• AES-128 uses a 256-bit key (32 bytes) which occupies one Efuse key block.
• AES-256 uses a 512-bit key (64 bytes) which occupies two Efuse key blocks.

This setting is ignored if either type of key is already burned to Efuse before the first boot. In this case, the pre-burned key is used and no new key is generated.

**Available options:**

• AES-128 key derived from 128 bits (SHA256(128 bits)) (SE-CURE_FLASH_ENCRYPTION_AES128_DERIVED)
• AES-128 (256-bit key) (SECURE_FLASH_ENCRYPTION_AES128)
• AES-256 (512-bit key) (SECURE_FLASH_ENCRYPTION_AES256)

**CONFIG_SECURE_FLASH_ENCRYPTION_MODE**

Enable usage mode

*Found in: Security features > CONFIG_SECURE_FLASH_ENC_ENABLED*

By default Development mode is enabled which allows ROM download mode to perform flash encryption operations (plaintext is sent to the device, and it encrypts it internally and writes ciphertext to flash.) This mode is not secure, it’s possible for an attacker to write their own chosen plaintext to flash.

Release mode should always be selected for production or manufacturing. Once enabled it’s no longer possible for the device in ROM Download Mode to use the flash encryption hardware.

Refer to the Flash Encryption section of the ESP-IDF Programmer’s Guide for details.

**Available options:**

• Development (NOT SECURE) (SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT)
• Release (SECURE_FLASH_ENCRYPTION_MODE_RELEASE)

**Potentially insecure options**

Contains:

• `CONFIG_SECURE_BOOT_V2_ALLOW_EFUSE_RD_DIS`
• `CONFIG_SECURE_BOOT_ALLOW_SHORT_APP_PARTITION`
• `CONFIG_SECURE_BOOT_ALLOW_JTAG`
• `CONFIG_SECURE_BOOT_ALLOW_ROM_BASIC`
Chapter 2. API Reference

- CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_DEC
- CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC
- CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE
- CONFIG_SECURE_BOOT_ALLOW_UNUSED_DIGEST_SLOTS
- CONFIG_SECURE_FLASH_REQUIRE_ALREADY_ENABLED

### CONFIG_SECURE_BOOT_ALLOW_ROM_BASIC

Leave ROM BASIC Interpreter available on reset

*Found in: Security features > Potentially insecure options*

By default, the BASIC ROM Console starts on reset if no valid bootloader is read from the flash.

When either flash encryption or secure boot are enabled, the default is to disable this BASIC fallback mode permanently via eFuse.

If this option is set, this eFuse is not burned and the BASIC ROM Console may remain accessible. Only set this option in testing environments.

*Default value:*
- No (disabled) if CONFIG_SECURE_BOOT_INSECURE || SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT

### CONFIG_SECURE_BOOT.Allow_JTAG

Allow JTAG Debugging

*Found in: Security features > Potentially insecure options*

If not set (default), the bootloader will permanently disable JTAG (across entire chip) on first boot when either secure boot or flash encryption is enabled.

Setting this option leaves JTAG on for debugging, which negates all protections of flash encryption and some of the protections of secure boot.

Only set this option in testing environments.

*Default value:*
- No (disabled) if CONFIG_SECURE_BOOT_INSECURE || SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT

### CONFIG_SECURE_BOOT_ALLOW_SHORT_APP_PARTITION

Allow app partition length not 64KB aligned

*Found in: Security features > Potentially insecure options*

If not set (default), app partition size must be a multiple of 64KB. App images are padded to 64KB length, and the bootloader checks any trailing bytes after the signature (before the next 64KB boundary) have not been written. This is because flash cache maps entire 64KB pages into the address space. This prevents an attacker from appending unverified data after the app image in the flash, causing it to be mapped into the address space.

Setting this option allows the app partition length to be unaligned, and disables padding of the app image to this length. It is generally not recommended to set this option, unless you have a legacy partitioning scheme which doesn’t support 64KB aligned partition lengths.

### CONFIG_SECURE_BOOT_V2_ALLOW_EFUSE_RD_DIS

Allow additional read protecting of efuses

*Found in: Security features > Potentially insecure options*
If not set (default, recommended), on first boot the bootloader will burn the WR_DIS_RD_DIS efuse when Secure Boot is enabled. This prevents any more efuses from being read protected.

If this option is set, it will remain possible to write the EFUSE_RD_DIS efuse field after Secure Boot is enabled. This may allow an attacker to read-protect the BLK2 efuse (for ESP32) and BLOCK4-BLOCK10 (i.e. BLOCK_KEY0-BLOCK_KEY5) (for other chips) holding the public key digest, causing an immediate denial of service and possibly allowing an additional fault injection attack to bypass the signature protection.

NOTE: Once a BLOCK is read-protected, the application will read all zeros from that block

NOTE: If UART ROM download mode (Permanently disabled (recommended)) or UART ROM download mode (Permanently switch to Secure mode (recommended)) is set, then it is _NOT_ possible to read/write efuses using espefuse.py utility. However, efuse can be read/written from the application

**CONFIG_SECURE_BOOT_ALLOW_UNUSED_DIGEST_SLOTS**

Leave unused digest slots available (not revoke)

*Found in: Security features > Potentially insecure options*

If not set (default), during startup in the app all unused digest slots will be revoked. To revoke unused slot will be called esp_efuse_set_digest_revoke(num_digest) for each digest. Revoking unused digest slots makes ensures that no trusted keys can be added later by an attacker. If set, it means that you have a plan to use unused digests slots later.

**Default value:**

- No (disabled) if `CONFIG_SECURE_BOOT_INSECURE` && `SOC_EFUSE_REVOKE_BOOT_KEY_DIGESTS`

**CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC**

Leave UART bootloader encryption enabled

*Found in: Security features > Potentially insecure options*

If not set (default), the bootloader will permanently disable UART bootloader encryption access on first boot. If set, the UART bootloader will still be able to access hardware encryption.

It is recommended to only set this option in testing environments.

**Default value:**

- No (disabled) if `SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT`

**CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_DEC**

Leave UART bootloader decryption enabled

*Found in: Security features > Potentially insecure options*

If not set (default), the bootloader will permanently disable UART bootloader decryption access on first boot. If set, the UART bootloader will still be able to access hardware decryption.

Only set this option in testing environments. Setting this option allows complete bypass of flash encryption.

**Default value:**

- No (disabled) if `SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT`
**CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE**

Leave UART bootloader flash cache enabled

*Found in: Security features > Potentially insecure options*

If not set (default), the bootloader will permanently disable UART bootloader flash cache access on first boot. If set, the UART bootloader will still be able to access the flash cache.

Only set this option in testing environments.

**Default value:**
- No (disabled) if `SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT`

**CONFIG_SECURE_FLASH_REQUIRE_ALREADY_ENABLED**

Require flash encryption to be already enabled

*Found in: Security features > Potentially insecure options*

If not set (default), and flash encryption is not yet enabled in eFuses, the 2nd stage bootloader will enable flash encryption: generate the flash encryption key and program eFuses. If this option is set, and flash encryption is not yet enabled, the bootloader will error out and reboot. If flash encryption is enabled in eFuses, this option does not change the bootloader behavior.

Only use this option in testing environments, to avoid accidentally enabling flash encryption on the wrong device. The device needs to have flash encryption already enabled using espfuse.py.

**Default value:**
- No (disabled) if `SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT`

**CONFIG_SECURE_FLASH_CHECK_ENC_EN_IN_APP**

Check Flash Encryption enabled on app startup

*Found in: Security features*

If set (default), in an app during startup code, there is a check of the flash encryption eFuse bit is on (as the bootloader should already have set it). The app requires this bit is on to continue work otherwise abort.

If not set, the app does not care if the flash encryption eFuse bit is set or not.

**Default value:**
- Yes (enabled) if `CONFIG_SECURE_FLASH_ENC_ENABLED`

**CONFIG_SECURE_UART_ROM_DL_MODE**

UART ROM download mode

*Found in: Security features*

**Available options:**
- UART ROM download mode (Permanently disabled (recommended)) (SECURE_DISABLE_ROM_DL_MODE)

If set, during startup the app will burn an eFuse bit to permanently disable the UART ROM Download Mode. This prevents any future use of esptool.py, espfuse.py and similar tools. Once disabled, if the SoC is booted with strapping pins set for ROM Download Mode then an error is printed instead.

It is recommended to enable this option in any production application where Flash Encryption and/or Secure Boot is enabled and access to Download Mode is not required.

It is also possible to permanently disable Download Mode by calling `esp_efuse_disable_rom_download_mode()` at runtime.
• UART ROM download mode (Permanently switch to Secure mode (recommended)) (SECURE_ENABLE_SECURE_ROM_DL_MODE)

If set, during startup the app will burn an eFuse bit to permanently switch the UART ROM Download Mode into a separate Secure Download mode. This option can only work if Download Mode is not already disabled by eFuse.

Secure Download mode limits the use of Download Mode functions to simple flash read, write and erase operations, plus a command to return a summary of currently enabled security features.

Secure Download mode is not compatible with the esptool.py flasher stub feature, espefuse.py, read/writing memory or registers, encrypted download, or any other features that interact with unsupported Download Mode commands.

Secure Download mode should be enabled in any application where Flash Encryption and/or Secure Boot is enabled. Disabling this option does not immediately cancel the benefits of the security features, but it increases the potential “attack surface” for an attacker to try and bypass them with a successful physical attack.

It is also possible to enable secure download mode at runtime by calling esp_efuse_enable_rom_secure_download_mode()

Note: Secure Download mode is not available for ESP32 (includes revisions till ECO3).

• UART ROM download mode (Enabled (not recommended)) (SECURE_INSECURE_ALLOW_DL_MODE)

This is a potentially insecure option. Enabling this option will allow the full UART download mode to stay enabled. This option SHOULD NOT BE ENABLED for production use cases.

Application manager

Contains:

• CONFIG_APP_EXCLUDE_PROJECT_NAME_VAR
• CONFIG_APP_EXCLUDE_PROJECT_VER_VAR
• CONFIG_APP_PROJECT_VER_FROM_CONFIG
• CONFIG_APP_RETRIEVE_LEN_ELF_SHA
• CONFIG_APP_COMPILE_TIME_DATE

CONFIG_APP_COMPILE_TIME_DATE

Use time/date stamp for app

*Found in: Application manager*

If set, then the app will be built with the current time/date stamp. It is stored in the app description structure. If not set, time/date stamp will be excluded from app image. This can be useful for getting the same binary image files made from the same source, but at different times.

*Default value:*

• Yes (enabled)

CONFIG_APP_EXCLUDE_PROJECT_VER_VAR

Exclude PROJECT_VER from firmware image

*Found in: Application manager*

The PROJECT_VER variable from the build system will not affect the firmware image. This value will not be contained in the esp_app_desc structure.

*Default value:*

• No (disabled)
**CONFIG_APP_EXCLUDE_PROJECT_NAME_VAR**

Exclude `PROJECT_NAME` from firmware image

*Found in: Application manager*

The `PROJECT_NAME` variable from the build system will not affect the firmware image. This value will not be contained in the `esp_app_desc` structure.

*Default value:*
- No (disabled)

**CONFIG_APP_PROJECT_VER_FROM_CONFIG**

Get the project version from Kconfig

*Found in: Application manager*

If this is enabled, then config item `APP_PROJECT_VER` will be used for the variable `PROJECT_VER`. Other ways to set `PROJECT_VER` will be ignored.

*Default value:*
- No (disabled)

**CONFIG_APP_PROJECT_VER**

Project version

*Found in: Application manager > CONFIG_APP_PROJECT_VER_FROM_CONFIG*

Project version

*Default value:*
- 1 if `CONFIG_APP_PROJECT_VER_FROM_CONFIG`

**CONFIG_APP_RETRIEVE_LEN_ELF_SHA**

The length of APP ELF SHA is stored in RAM(chars)

*Found in: Application manager*

At startup, the app will read this many hex characters from the embedded APP ELF SHA-256 hash value and store it in static RAM. This ensures the app ELF SHA-256 value is always available if it needs to be printed by the panic handler code. Changing this value will change the size of a static buffer, in bytes.

*Range:*
- from 8 to 64

*Default value:*
- 16

**Serial flasher config**

Contains:

- `CONFIG_ESPTOOLPY_AFTER`
- `CONFIG_ESPTOOLPY_BEFORE`
- `CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE`
- `CONFIG_ESPTOOLPY_NO_STUB`
- `CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE`
- `CONFIG_ESPTOOLPY_FLASHSIZE`
- `CONFIG_ESPTOOLPY_FLASHMODE`
- `CONFIG_ESPTOOLPY_FLASHFREQ`
Chapter 2. API 参考

**CONFIG_ESPTOOLPY_NO_STUB**

Disable download stub

*Found in: Serial flasher config*

The flasher tool sends a precompiled download stub first by default. That stub allows things like compressed downloads and more. Usually you should not need to disable that feature

*Default value:*

- No (disabled)

**CONFIG_ESPTOOLPY_FLASHMODE**

Flash SPI mode

*Found in: Serial flasher config*

Mode the flash chip is flashed in, as well as the default mode for the binary to run in.

*Available options:*

- QIO (ESPTOOLPY_FLASHMODE_QIO)
- QOUT (ESPTOOLPY_FLASHMODE_QOUT)
- DIO (ESPTOOLPY_FLASHMODE_DIO)
- DOUT (ESPTOOLPY_FLASHMODE_DOUT)
- OPI (ESPTOOLPY_FLASHMODE_OPI)

**CONFIG_ESPTOOLPY_FLASH_SAMPLE_MODE**

Flash Sampling Mode

*Found in: Serial flasher config*

*Available options:*

- STR Mode (ESPTOOLPY_FLASH_SAMPLE_MODE_STR)
- DTR Mode (ESPTOOLPY_FLASH_SAMPLE_MODE_DTR)

**CONFIG_ESPTOOLPY_FLASHFREQ**

Flash SPI speed

*Found in: Serial flasher config*

*Available options:*

- 120 MHz (ESPTOOLPY_FLASHFREQ_120M)
- 80 MHz (ESPTOOLPY_FLASHFREQ_80M)
- 60 MHz (ESPTOOLPY_FLASHFREQ_60M)
- 48 MHz (ESPTOOLPY_FLASHFREQ_48M)
- 40 MHz (ESPTOOLPY_FLASHFREQ_40M)
- 30 MHz (ESPTOOLPY_FLASHFREQ_30M)
- 26 MHz (ESPTOOLPY_FLASHFREQ_26M)
- 24 MHz (ESPTOOLPY_FLASHFREQ_24M)
- 20 MHz (ESPTOOLPY_FLASHFREQ_20M)
- 15 MHz (ESPTOOLPY_FLASHFREQ_15M)

**CONFIG_ESPTOOLPY_FLASHSIZE**

Flash size

*Found in: Serial flasher config*

SPI flash size, in megabytes

*Available options:*

- 1347
Chapter 2. API

- 1 MB (ESPTOOLPY_FLASHSIZE_1MB)
- 2 MB (ESPTOOLPY_FLASHSIZE_2MB)
- 4 MB (ESPTOOLPY_FLASHSIZE_4MB)
- 8 MB (ESPTOOLPY_FLASHSIZE_8MB)
- 16 MB (ESPTOOLPY_FLASHSIZE_16MB)
- 32 MB (ESPTOOLPY_FLASHSIZE_32MB)
- 64 MB (ESPTOOLPY_FLASHSIZE_64MB)
- 128 MB (ESPTOOLPY_FLASHSIZE_128MB)

**CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE**

Detect flash size when flashing bootloader

*Found in: Serial flasher config*

If this option is set, flashing the project will automatically detect the flash size of the target chip and update the bootloader image before it is flashed.

Enabling this option turns off the image protection against corruption by a SHA256 digest. Updating the bootloader image before flashing would invalidate the digest.

**Default value:**
- No (disabled)

**CONFIG_ESPTOOLPY_BEFORE**

Before flashing

*Found in: Serial flasher config*

Configure whether esptool.py should reset the ESP32 before flashing.

Automatic resetting depends on the RTS & DTR signals being wired from the serial port to the ESP32. Most USB development boards do this internally.

**Available options:**
- Reset to bootloader (ESPTOOLPY_BEFORE_RESET)
- No reset (ESPTOOLPY_BEFORE_NORESET)

**CONFIG_ESPTOOLPY_AFTER**

After flashing

*Found in: Serial flasher config*

Configure whether esptool.py should reset the ESP32 after flashing.

Automatic resetting depends on the RTS & DTR signals being wired from the serial port to the ESP32. Most USB development boards do this internally.

**Available options:**
- Reset after flashing (ESPTOOLPY_AFTER_RESET)
- Stay in bootloader (ESPTOOLPY_AFTER_NORESET)

**Partition Table**

Contains:

- `CONFIG_PARTITION_TABLE_CUSTOM_FILENAME`
- `CONFIG_PARTITION_TABLE_MD5`
- `CONFIG_PARTITION_TABLE_OFFSET`
- `CONFIG_PARTITION_TABLE_TYPE`
CONFIG_PARTITION_TABLE_TYPE

Partition Table

Found in: Partition Table

The partition table to flash to the ESP32. The partition table determines where apps, data and other resources are expected to be found.

The predefined partition table CSV descriptions can be found in the components/partition_table directory. These are mostly intended for example and development use, it’s expect that for production use you will copy one of these CSV files and create a custom partition CSV for your application.

Available options:
• Single factory app, no OTA (PARTITION_TABLE_SINGLE_APP)
  This is the default partition table, designed to fit into a 2MB or larger flash with a single 1MB app partition.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp.csv
  This partition table is not suitable for an app that needs OTA (over the air update) capability.
• Single factory app (large), no OTA (PARTITION_TABLE_SINGLE_APP_LARGE)
  This is a variation of the default partition table, that expands the 1MB app partition size to 1.5MB to fit more code.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp_large.csv
  This partition table is not suitable for an app that needs OTA (over the air update) capability.
• Factory app, two OTA definitions (PARTITION_TABLE_TWO_OTA)
  This is a basic OTA-enabled partition table with a factory app partition plus two OTA app partitions. All are 1MB, so this partition table requires 4MB or larger flash size.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_two_ota.csv
• Custom partition table CSV (PARTITION_TABLE_CUSTOM)
  Specify the path to the partition table CSV to use for your project.
  Consult the Partition Table section in the ESP-IDF Programmers Guide for more information.
• Single factory app, no OTA, encrypted NVS (PARTITION_TABLE_SINGLE_APP_ENCRYPTED_NVS)
  This is a variation of the default “Single factory app, no OTA” partition table that supports encrypted NVS when using flash encryption. See the Flash Encryption section in the ESP-IDF Programmers Guide for more information.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp_encr_nvs.csv
• Single factory app (large), no OTA, encrypted NVS (PARTITION_TABLE_SINGLE_APP_LARGE_ENC_NVS)
  This is a variation of the “Single factory app (large), no OTA” partition table that supports encrypted NVS when using flash encryption. See the Flash Encryption section in the ESP-IDF Programmers Guide for more information.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_singleapp_large_encr_nvs.csv
• Factory app, two OTA definitions, encrypted NVS (PARTITION_TABLE_TWO_OTA_ENCRYPTED_NVS)
  This is a variation of the “Factory app, two OTA definitions” partition table that supports encrypted NVS when using flash encryption. See the Flash Encryption section in the ESP-IDF Programmers Guide for more information.
  The corresponding CSV file in the IDF directory is components/partition_table/partitions_two_ota_encr_nvs.csv
Name of the custom partition CSV filename. This path is evaluated relative to the project root directory.

**Default value:**
- “partitions.csv”

**CONFIG_PARTITION_TABLE_OFFSET**

Offset of partition table

*Found in: Partition Table*

The address of partition table (by default 0x8000). Allows you to move the partition table, it gives more space for the bootloader. Note that the bootloader and app will both need to be compiled with the same PARTITION_TABLE_OFFSET value.

This number should be a multiple of 0x1000.

Note that partition offsets in the partition table CSV file may need to be changed if this value is set to a higher value. To have each partition offset adapt to the configured partition table offset, leave all partition offsets blank in the CSV file.

**Default value:**
- “0x8000”

**CONFIG_PARTITION_TABLE_MD5**

Generate an MD5 checksum for the partition table

*Found in: Partition Table*

Generate an MD5 checksum for the partition table for protecting the integrity of the table. The generation should be turned off for legacy bootloaders which cannot recognize the MD5 checksum in the partition table.

**Default value:**
- Yes (enabled) if `CONFIG_APP_COMPATIBLE_PRE_V3_1_BOOTLOADERS`

**Compiler options**

Contains:

- `CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL`
- `CONFIG_COMPILER_FLOAT_LIB_FROM`
- `CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT`
- `CONFIG_COMPILER_DUMP_RTL_FILES`
- `CONFIG_COMPILER_WARN_WRITE_STRINGS`
- `CONFIG_COMPILER_CXX_EXCEPTIONS`
- `CONFIG_COMPILER_CXX_RTTI`
- `CONFIG_COMPILER_OPTIMIZATION`
- `CONFIG_COMPILER_HIDE_PATHS_MACROS`
- `CONFIG_COMPILER_STACK_CHECK_MODE`

**CONFIG_COMPILER_OPTIMIZATION**

Optimization Level

*Found in: Compiler options*

This option sets compiler optimization level (gcc -O argument) for the app.

- The “Default” setting will add the -Og flag to CFLAGS.
- The “Size” setting will add the -Os flag to CFLAGS.
- The “Performance” setting will add the -O2 flag to CFLAGS.
• The “None” setting will add the -O0 flag to CFLAGS.

The “Size” setting cause the compiled code to be smaller and faster, but may lead to difficulties of correlating code addresses to source file lines when debugging.

The “Performance” setting causes the compiled code to be larger and faster, but will be easier to correlated code addresses to source file lines.

“None” with -O0 produces compiled code without optimization.

Note that custom optimization levels may be unsupported.

Compiler optimization for the IDF bootloader is set separately, see the BOOT-LOADER_COMPILER_OPTIMIZATION setting.

Available options:
• Debug (-Og) (COMPILER_OPTIMIZATION_DEFAULT)
• Optimize for size (-Os) (COMPILER_OPTIMIZATION_SIZE)
• Optimize for performance (-O2) (COMPILER_OPTIMIZATION_PERF)
• Debug without optimization (-O0) (COMPILER_OPTIMIZATION_NONE)

CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL

Assertion level

Found in: Compiler options

Assertions can be:
• Enabled. Failure will print verbose assertion details. This is the default.
• Set to “silent” to save code size (failed assertions will abort() but user needs to use the aborting address to find the line number with the failed assertion.)
• Disabled entirely (not recommended for most configurations.) -DNDEBUG is added to CPPFLAGS in this case.

Available options:
• Enabled (COMPILER_OPTIMIZATION_ASSERTIONS_ENABLE)
  Enable assertions. Assertion content and line number will be printed on failure.
• Silent (saves code size) (COMPILER_OPTIMIZATION_ASSERTIONS_SILENT)
  Enable silent assertions. Failed assertions will abort(), user needs to use the aborting address to find the line number with the failed assertion.
• Disabled (sets -DNDEBUG) (COMPILER_OPTIMIZATION_ASSERTIONS_DISABLE)
  If assertions are disabled, -DNDEBUG is added to CPPFLAGS.

CONFIG_COMPILER_FLOAT_LIB_FROM

Compiler float lib source

Found in: Compiler options

In the soft-fp part of libgcc, riscv version is written in C, and handles all edge cases in IEEE754, which makes it larger and performance is slow.

RVfplib is an optimized RISC-V library for FP arithmetic on 32-bit integer processors, for single and double-precision FP. RVfplib is “fast”, but it has a few exceptions from IEEE 754 compliance.

Available options:
• libgcc (COMPILER_FLOAT_LIB_FROM_GCCLIB)
• librvp (COMPILER_FLOAT_LIB_FROM_RVFPLIB)

CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT
Disable messages in ESP_RETURN_ON_* and ESP_EXIT_ON_* macros

*Found in: Compiler options*

If enabled, the error messages will be discarded in following check macros:
- ESP_RETURN_ON_ERROR
- ESP_EXIT_ON_ERROR
- ESP_RETURN_ON_FALSE
- ESP_EXIT_ON_FALSE

*Default value:*
- No (disabled)

**CONFIG_COMPILER_HIDE_PATHS_MACROS**

Replace ESP-IDF and project paths in binaries

*Found in: Compiler options*

When expanding the __FILE__ and __BASE_FILE__ macros, replace paths inside ESP-IDF with paths relative to the placeholder string “IDF”, and convert paths inside the project directory to relative paths.

This allows building the project with assertions or other code that embeds file paths, without the binary containing the exact path to the IDF or project directories.

This option passes -fmacro-prefix-map options to the GCC command line. To replace additional paths in your binaries, modify the project CMakeLists.txt file to pass custom -fmacro-prefix-map or -ffile-prefix-map arguments.

*Default value:*
- Yes (enabled)

**CONFIG_COMPILER_CXX_EXCEPTIONS**

Enable C++ exceptions

*Found in: Compiler options*

Enabling this option compiles all IDF C++ files with exception support enabled.

Disabling this option disables C++ exception support in all compiled files, and any libstdc++ code which throws an exception will abort instead.

Enabling this option currently adds an additional ~500 bytes of heap overhead when an exception is thrown in user code for the first time.

*Default value:*
- No (disabled)

Contains:
- **CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE**

**CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE**

Emergency Pool Size

*Found in: Compiler options > CONFIG_COMPILER_CXX_EXCEPTIONS*

Size (in bytes) of the emergency memory pool for C++ exceptions. This pool will be used to allocate memory for thrown exceptions when there is not enough memory on the heap.

*Default value:*
- 0 if CONFIG_COMPILER_CXX_EXCEPTIONS
**CONFIG_COMPILER_CXX_RTTI**

Enable C++ run-time type info (RTTI)

*Found in: Compiler options*

Enabling this option compiles all C++ files with RTTI support enabled. This increases binary size (typically by tens of kB) but allows using dynamic_cast conversion and typeid operator.

*Default value:*

- No (disabled)

**CONFIG_COMPILER_STACK_CHECK_MODE**

Stack smashing protection mode

*Found in: Compiler options*

Stack smashing protection mode. Emit extra code to check for buffer overflows, such as stack smashing attacks. This is done by adding a guard variable to functions with vulnerable objects. The guards are initialized when a function is entered and then checked when the function exits. If a guard check fails, program is halted. Protection has the following modes:

- In NORMAL mode (GCC flag: -fstack-protector) only functions that call alloca, and functions with buffers larger than 8 bytes are protected.
- STRONG mode (GCC flag: -fstack-protector-strong) is like NORMAL, but includes additional functions to be protected - those that have local array definitions, or have references to local frame addresses.
- In OVERALL mode (GCC flag: -fstack-protector-all) all functions are protected.

Modes have the following impact on code performance and coverage:

- performance: NORMAL > STRONG > OVERALL
- coverage: NORMAL < STRONG < OVERALL

The performance impact includes increasing the amount of stack memory required for each task.

*Available options:*

- None (COMPILER_STACK_CHECK_MODE_NONE)
- Normal (COMPILER_STACK_CHECK_MODE_NORM)
- Strong (COMPILER_STACK_CHECK_MODE_STRONG)
- Overall (COMPILER_STACK_CHECK_MODE_ALL)

**CONFIG_COMPILER_WARN_WRITE_STRINGS**

Enable -Wwrite-strings warning flag

*Found in: Compiler options*

Adds -Wwrite-strings flag for the C/C++ compilers.

For C, this gives string constants the type `const char[]` so that copying the address of one into a non-const `char *` pointer produces a warning. This warning helps to find at compile time code that tries to write into a string constant.

For C++, this warns about the deprecated conversion from string literals to `char *`.

*Default value:*

- No (disabled)

**CONFIG_COMPILER_DUMP_RTL_FILES**

Dump RTL files during compilation

*Found in: Compiler options*
If enabled, RTL files will be produced during compilation. These files can be used by other tools, for example to calculate call graphs.

**Component config**

Contains:

- ADC and ADC Calibration
- Application Level Tracing
- Bluetooth
- Common ESP-related
- Core dump
- Driver Configurations
- eFuse Bit Manager
- CONFIG_BLE_MESH
- ESP HTTP client
- ESP HTTPS OTA
- ESP HTTPS server
- ESP NETIF Adapter
- ESP PSRAM
- ESP Ringbuf
- ESP System Settings
- ESP-MQTT Configurations
- ESP-TLS
- Ethernet
- Event Loop Library
- FAT Filesystem support
- FreeRTOS
- GDB Stub
- Hardware Abstraction Layer (HAL) and Low Level (LL)
- Hardware Settings
- Heap memory debugging
- High resolution timer (esp_timer)
- HTTP Server
- IPC (Inter-Processor Call)
- LCD and Touch Panel
- Log output
- LWIP
- mbedTLS
- Newlib
- NVS
- OpenThread
- PHY
- Power Management
- Protocomm
- PThreads
- SPI Flash driver
- SPIFFS Configuration
- Supplicant
- TCP Transport
- Ultra Low Power (ULP) Co-processor
- Unity unit testing library
- Virtual file system
- Wear Levelling
- Wi-Fi
- Wi-Fi Provisioning Manager
Application Level Tracing Contains:

- `CONFIG_APPTTRACE_DESTINATION1`
- `CONFIG_APPTTRACE_DESTINATION2`
- FreeRTOS SystemView Tracing
- `CONFIG_APPTTRACE_GCOV_ENABLE`
- `CONFIG_APPTTRACE_BUF_SIZE`
- `CONFIG_APPTTRACE_PENDING_DATA_SIZE_MAX`
- `CONFIG_APPTTRACE_POSTMORTEM_FLUSH_THRESH`
- `CONFIG_APPTTRACE_ONPANIC_HOST_FLUSH_TMO`
- `CONFIG_APPTTRACE_UART_BAUDRATE`
- `CONFIG_APPTTRACE_UART_RX_GPIO`
- `CONFIG_APPTTRACE_UART_RX_BUFF_SIZE`
- `CONFIG_APPTTRACE_UART_TASK_PRIO`
- `CONFIG_APPTTRACE_UART_TX_MSG_SIZE`
- `CONFIG_APPTTRACE_UART_TX_GPIO`
- `CONFIG_APPTTRACE_UART_TX_BUFF_SIZE`

**CONFIG_APPTTRACE_DESTINATION1**

Data Destination 1

*Found in:* Component config > Application Level Tracing

Select destination for application trace: JTAG or none (to disable).

*Available options:*
- JTAG (`APPTTRACE_DEST_JTAG`)
- None (`APPTTRACE_DEST_NONE`)

**CONFIG_APPTTRACE_DESTINATION2**

Data Destination 2

*Found in:* Component config > Application Level Tracing

Select destination for application trace: UART(XX) or none (to disable).

*Available options:*
- UART0 (`APPTTRACE_DEST_UART0`)
- UART1 (`APPTTRACE_DEST_UART1`)
- UART2 (`APPTTRACE_DEST_UART2`)
- USB_CDC (`APPTTRACE_DEST_USB_CDC`)
- None (`APPTTRACE_DEST_UART_NONE`)

**CONFIG_APPTTRACE_UART_TX_GPIO**

UART TX on GPIO#

*Found in:* Component config > Application Level Tracing

This GPIO is used for UART TX pin.

**CONFIG_APPTTRACE_UART_RX_GPIO**

UART RX on GPIO#

*Found in:* Component config > Application Level Tracing

This GPIO is used for UART RX pin.
**CONFIG_APPTRACE_UART_BAUDRATE**

UART baud rate

*Found in: Component config > Application Level Tracing*

This baud rate is used for UART.

The app’s maximum baud rate depends on the UART clock source. If Power Management is disabled, the UART clock source is the APB clock and all baud rates in the available range will be sufficiently accurate. If Power Management is enabled, REF_TICK clock source is used so the baud rate is divided from 1MHz. Baud rates above 1Mbps are not possible and values between 500Kbps and 1Mbps may not be accurate.

**CONFIG_APPTRACE_UART_RX_BUFF_SIZE**

UART RX ring buffer size

*Found in: Component config > Application Level Tracing*

Size of the UART input ring buffer. This size related to the baudrate, system tick frequency and amount of data to transfer. The data placed to this buffer before sent out to the interface.

**CONFIG_APPTRACE_UART_TX_BUFF_SIZE**

UART TX ring buffer size

*Found in: Component config > Application Level Tracing*

Size of the UART output ring buffer. This size related to the baudrate, system tick frequency and amount of data to transfer.

**CONFIG_APPTRACE_UART_TX_MSG_SIZE**

UART TX message size

*Found in: Component config > Application Level Tracing*

Maximum size of the single message to transfer.

**CONFIG_APPTRACE_UART_TASK_PRIO**

UART Task Priority

*Found in: Component config > Application Level Tracing*

UART task priority. In case of high events rate, this parameter could be changed up to (config-MAX_PRIORITIES-1).

*Range:*
- from 1 to 32

*Default value:*
- 1

**CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO**

Timeout for flushing last trace data to host on panic

*Found in: Component config > Application Level Tracing*

Timeout for flushing last trace data to host in case of panic. In ms. Use -1 to disable timeout and wait forever.
**CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH**

Threshold for flushing last trace data to host on panic

*Found in: Component config > Application Level Tracing*

Threshold for flushing last trace data to host on panic in post-mortem mode. This is minimal amount of data needed to perform flush. In bytes.

**CONFIG_APPTRACE_BUF_SIZE**

Size of the apptrace buffer

*Found in: Component config > Application Level Tracing*

Size of the memory buffer for trace data in bytes.

**CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX**

Size of the pending data buffer

*Found in: Component config > Application Level Tracing*

Size of the buffer for events in bytes. It is useful for buffering events from the time critical code (scheduler, ISRs etc). If this parameter is 0 then events will be discarded when main HW buffer is full.

**FreeRTOS SystemView Tracing** Contains:

- `CONFIG_APPTRACE_SV_CPU`
- `CONFIG_APPTRACE_SV_EVT_ISR_ENTER_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_ISR_EXIT_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_ISR_TO_SCHED_ENABLE`
- `CONFIG_APPTRACE_SV_MAX_TASKS`
- `CONFIG_APPTRACE_SV_EVT_IDLE_ENABLE`
- `CONFIG_APPTRACE_SV_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_TASK_CREATE_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_TASK_START_EXEC_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_TASK_START_READY_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_TASK_STOP_EXEC_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_TASK_STOP_READY_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_TIMER_ENTER_ENABLE`
- `CONFIG_APPTRACE_SV_EVT_TIMER_EXIT_ENABLE`
- `CONFIG_APPTRACE_SV_TS_SOURCE`
- `CONFIG_APPTRACE_SV_EVT_OVERFLOW_ENABLE`
- `CONFIG_APPTRACE_SV_BUF_WAIT_TMO`

**CONFIG_APPTRACE_SV_ENABLE**

SystemView Tracing Enable

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables support for SEGGER SystemView tracing functionality.

**CONFIG_APPTRACE_SV_DEST**

SystemView destination

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_APPTRACE_SV_ENABLE*

SystemView will transfer data through defined interface.
Available options:

- Data destination JTAG (APPTRACE_SV_DEST_JTAG)
  Send SEGGER SystemView events through JTAG interface.
- Data destination UART (APPTRACE_SV_DEST_UART)
  Send SEGGER SystemView events through UART interface.

CONFIG_APPTRACE_SV_CPU

CPU to trace

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Define the CPU to trace by SystemView.

Available options:

- CPU0 (APPTRACE_SV_DEST_CPU_0)
  Send SEGGER SystemView events for Pro CPU.
- CPU1 (APPTRACE_SV_DEST_CPU_1)
  Send SEGGER SystemView events for App CPU.

CONFIG_APPTRACE_SV_TS_SOURCE

Timer to use as timestamp source

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

SystemView needs to use a hardware timer as the source of timestamps when tracing. This option selects the timer for it.

Available options:

- CPU cycle counter (CCOUNT) (APPTRACE_SV_TS_SOURCE_CCOUNT)
- General Purpose Timer (Timer Group) (APPTRACE_SV_TS_SOURCE_GPTIMER)
- esp_timer high resolution timer (APPTRACE_SV_TS_SOURCE_ESP_TIMER)

CONFIG_APPTRACE_SV_MAX_TASKS

Maximum supported tasks

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Configures maximum supported tasks in sysview debug

CONFIG_APPTRACE_SV_BUF_WAIT_TMO

Trace buffer wait timeout

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Configures timeout (in us) to wait for free space in trace buffer. Set to -1 to wait forever and avoid lost events.

CONFIG_APPTRACE_SV_EVT_OVERFLOW_ENABLE

Trace Buffer Overflow Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Trace Buffer Overflow” event.
Chapter 2. API 参考

**CONFIG_APPTRACE_SV_EVT_ISR_ENTER_ENABLE**
ISR Enter Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “ISR Enter” event.

**CONFIG_APPTRACE_SV_EVT_ISR_EXIT_ENABLE**
ISR Exit Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “ISR Exit” event.

**CONFIG_APPTRACE_SV_EVT_ISR_TO_SCHED_ENABLE**
ISR Exit to Scheduler Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “ISR to Scheduler” event.

**CONFIG_APPTRACE_SV_EVT_TASK_START_EXEC_ENABLE**
Task Start Execution Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Task Start Execution” event.

**CONFIG_APPTRACE_SV_EVT_TASK_STOP_EXEC_ENABLE**
Task Stop Execution Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Task Stop Execution” event.

**CONFIG_APPTRACE_SV_EVT_TASK_START_READY_ENABLE**
Task Start Ready State Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Task Start Ready State” event.

**CONFIG_APPTRACE_SV_EVT_TASK_STOP_READY_ENABLE**
Task Stop Ready State Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Task Stop Ready State” event.

**CONFIG_APPTRACE_SV_EVT_TASK_CREATE_ENABLE**
Task Create Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Task Create” event.
CONFIG_APPTRACE_SV_EVT_TASK_TERMINATE_ENABLE

Task Terminate Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Task Terminate” event.

CONFIG_APPTRACE_SV_EVT_IDLE_ENABLE

System Idle Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “System Idle” event.

CONFIG_APPTRACE_SV_EVT_TIMER_ENTER_ENABLE

Timer Enter Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Timer Enter” event.

CONFIG_APPTRACE_SV_EVT_TIMER_EXIT_ENABLE

Timer Exit Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables “Timer Exit” event.

CONFIG_APPTRACE_GCOV_ENABLE

GCOV to Host Enable

*Found in: Component config > Application Level Tracing*

Enables support for GCOV data transfer to host.

**Bluetooth** Contains:

- *Bluedroid Options*
- *CONFIG_BT_ENABLED*
- *Controller Options*
- *NimBLE Options*

CONFIG_BT_ENABLED

Bluetooth

*Found in: Component config > Bluetooth*

Select this option to enable Bluetooth and show the submenu with Bluetooth configuration choices.

CONFIG_BT_HOST

Host

*Found in: Component config > Bluetooth > CONFIG_BT_ENABLED*

This helps to choose Bluetooth host stack

**Available options:**
Chapter 2. API

- **Bluedroid - Dual-mode (BT_BLUEDROID_ENABLED)**
  This option is recommended for classic Bluetooth or for dual-mode use cases
- **NimBLE - BLE only (BT_NIMBLE_ENABLED)**
  This option is recommended for BLE only use cases to save on memory
- **Disabled (BT_CONTROLLER_ONLY)**
  This option is recommended when you want to communicate directly with the controller (without any host) or when you are using any other host stack not supported by Espressif (not mentioned here).

**CONFIG_BT_CONTROLLER**

Controller

*Found in: Component config > Bluetooth > CONFIG_BT_ENABLED*

This helps to choose Bluetooth controller stack

**Available options:**
- **Enabled (BT_CONTROLLER_ENABLED)**
  This option is recommended for Bluetooth controller use cases
- **Disabled (BT_CONTROLLER_DISABLED)**
  This option is recommended for Bluetooth Host only use cases

**Bluedroid Options**

Contains:

- **CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK**
- **CONFIG_BT_BLUEDROID_MEM_DEBUG**
- **CONFIG_BT_BTU_TASK_STACK_SIZE**
- **CONFIG_BT_BTC_TASK_STACK_SIZE**
- **CONFIG_BT_BLE_ENABLED**
- **BT_DEBUG LOG LEVEL**
- **CONFIG_BT_ACL_CONNECTIONS**
- **CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST**
- **CONFIG_BT_CLASSIC_ENABLED**
- **CONFIG_BT_HID_ENABLED**
- **CONFIG_BT_STACK_NO_LOG**
- **CONFIG_BT_BLE_42_FEATURES_SUPPORTED**
- **CONFIG_BT_BLE_50_FEATURES_SUPPORTED**
- **CONFIG_BT_MULTI_CONNECTION_ENABLE**
- **CONFIG_BT_MAX_DEVICE_NAME_LEN**
- **CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN**
- **CONFIG_BT_SSP_ENABLED**
- **CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE**
- **CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT**
- **CONFIG_BT_BLE_RPA_SUPPORTED**
- **CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY**
- **CONFIG_BT_HFP_WBS_ENABLE**

**CONFIG_BT_BTC_TASK_STACK_SIZE**

Bluetooth event (callback to application) task stack size

*Found in: Component config > Bluetooth > Bluedroid Options*

This select btc task stack size

**Default value:**
- 3072 if BT_BLUEDROID_ENABLED & BT_BLUEDROID_ENABLED
CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE

The cpu core which Bluedroid run

*Found in: Component config > Bluetooth > Bluedroid Options*

Which the cpu core to run Bluedroid. Can choose core0 and core1. Can not specify no-affinity.

**Available options:**
- Core 0 (PRO CPU) (BT_BLUEDROID_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (BT_BLUEDROID_PINNED_TO_CORE_1)

CONFIG_BT_BTU_TASK_STACK_SIZE

Bluetooth Bluedroid Host Stack task stack size

*Found in: Component config > Bluetooth > Bluedroid Options*

This select btu task stack size

**Default value:**
- 4096 if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_BLUEDROID_MEM_DEBUG

Bluedroid memory debug

*Found in: Component config > Bluetooth > Bluedroid Options*

Bluedroid memory debug

**Default value:**
- No (disabled) if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_CLASSIC_ENABLED

Classic Bluetooth

*Found in: Component config > Bluetooth > Bluedroid Options*

For now this option needs “SMP_ENABLE” to be set to yes

**Default value:**
- No (disabled) if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_A2DP_ENABLE

A2DP

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED*

Advanced Audio Distribution Profile

**Default value:**
- No (disabled) if CONFIG_BT_CLASSIC_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_SPP_ENABLED

SPP

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED*

This enables the Serial Port Profile

**Default value:**
- No (disabled) if CONFIG_BT_CLASSIC_ENABLED && BT_BLUEDROID_ENABLED
CONFIG_BT_L2CAP_ENABLED

BT L2CAP

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED*

This enables the Logical Link Control and Adaptation Layer Protocol. Only supported classic bluetooth.

*Default value:*

- No (disabled) if `CONFIG_BT_CLASSIC_ENABLED` && `BT_BLUEDROID_ENABLED`

CONFIG_BT_HFP_ENABLE

Hands Free/Handset Profile

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED*

*Default value:*

- No (disabled) if `CONFIG_BT_CLASSIC_ENABLED` && `BT_BLUEDROID_ENABLED`

CONFIG_BT_HFP_ROLE

Hands-free Profile Role configuration

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED > CONFIG_BT_HFP_ENABLE*

*Available options:*

- Hands Free Unit (BT_HFP_CLIENT_ENABLE)
- Audio Gateway (BT_HFP_AG_ENABLE)

CONFIG_BT_HFP_AUDIO_DATA_PATH

audio(SCO) data path

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED > CONFIG_BT_HFP_ENABLE*

SCO data path, i.e. HCI or PCM. This option is set using API “esp_bredr_sco_datapath_set” in Bluetooth host. Default SCO data path can also be set in Bluetooth Controller.

*Available options:*

- PCM (BT_HFP_AUDIO_DATA_PATH_PCM)
- HCI (BT_HFP_AUDIO_DATA_PATH_HCI)

CONFIG_BT_HFP_WBS_ENABLE

Wide Band Speech

*Found in: Component config > Bluetooth > Bluedroid Options*

This enables Wide Band Speech. Should disable it when SCO data path is PCM. Otherwise there will be no data transmitted via GPIOs.

*Default value:*

- Yes (enabled) if `BT_HFP_AUDIO_DATA_PATH_HCI` && `BT_HFP_AUDIO_DATA_PATH_PCM` && `BT_BLUEDROID_ENABLED`

CONFIG_BT_HID_ENABLED

Classic BT HID

*Found in: Component config > Bluetooth > Bluedroid Options*

This enables the BT HID Host
Default value:
- No (disabled) if `CONFIG_BT_CLASSIC_ENABLED` && `BT_BLUEDROID_ENABLED`

**CONFIG_BT_HID_ROLE**

Profile Role configuration

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_HID_ENABLED*

**Available options:**
- Classic BT HID Host (BT_HID_HOST_ENABLED)
  This enables the BT HID Host
- Classic BT HID Device (BT_HID_DEVICE_ENABLED)
  This enables the BT HID Device

**CONFIG_BT_SSP_ENABLED**

Secure Simple Pairing

*Found in: Component config > Bluetooth > Bluedroid Options*

This enables the Secure Simple Pairing. If disable this option, Bluedroid will only support Legacy Pairing

**Default value:**
- Yes (enabled) if `CONFIG_BT_CLASSIC_ENABLED` && `BT_BLUEDROID_ENABLED`

**CONFIG_BT_BLE_ENABLED**

Bluetooth Low Energy

*Found in: Component config > Bluetooth > Bluedroid Options*

This enables Bluetooth Low Energy

**Default value:**
- Yes (enabled) if `BT_BLUEDROID_ENABLED` && `BT_BLUEDROID_ENABLED`

**CONFIG_BT_GATTS_ENABLE**

Include GATT server module (GATTS)

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED*

This option can be disabled when the app work only on gatt client mode

**Default value:**
- Yes (enabled) if `CONFIG_BT_BLE_ENABLED` && `BT_BLUEDROID_ENABLED`

**CONFIG_BT_GATTS_PPCP_CHAR_GAP**

Enable Peripheral Preferred Connection Parameters characteristic in GAP service

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE*

This enables “Peripheral Preferred Connection Parameters” characteristic (UUID: 0x2A04) in GAP service that has connection parameters like min/max connection interval, slave latency and supervision timeout multiplier

**Default value:**
- No (disabled) if `CONFIG_BT_GATTS_ENABLE` && `BT_BLUEDROID_ENABLED`
**CONFIG_BT_BLE_BLUFI_ENABLE**

Include blufi function

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE*

This option can be close when the app does not require blufi function.

**Default value:**
- No (disabled) if CONFIG_BT_GATTS_ENABLE && BT_BLUEDROID_ENABLED

**CONFIG_BT_GATT_MAX_SR_PROFILES**

Max GATT Server Profiles

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE*

Maximum GATT Server Profiles Count

**Range:**
- from 1 to 32 if CONFIG_BT_GATTS_ENABLE && BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

**Default value:**
- 8 if CONFIG_BT_GATTS_ENABLE && BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

**CONFIG_BT_GATT_MAX_SR_ATTRIBUTES**

Max GATT Service Attributes

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE*

Maximum GATT Service Attributes Count

**Range:**
- from 1 to 500 if CONFIG_BT_GATTS_ENABLE && BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

**Default value:**
- 100 if CONFIG_BT_GATTS_ENABLE && BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

**CONFIG_BT_GATTS_SEND_SERVICE_CHANGE_MODE**

GATTS Service Change Mode

*Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE*

Service change indication mode for GATT Server.

**Available options:**
- GATTS manually send service change indication (BT_GATTS_SEND_SERVICE_CHANGE_MANUAL)
  Manually send service change indication through API esp_ble_gatts_send_service_change_indication()  
- GATTS automatically send service change indication (BT_GATTS_SEND_SERVICE_CHANGE_AUTO)
  Let Bluedroid handle the service change indication internally
CONFIG_BT_GATTC_ENABLE

Include GATT client module(GATTC)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be close when the app work only on gatt server mode

Default value:
• Yes (enabled) if CONFIG_BT_BLE_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_GATTC_MAX_CACHE_CHAR

Max gattc cache characteristic for discover

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

Maximum GATTC cache characteristic count

Range:
• from 1 to 500 if CONFIG_BT_GATTC_ENABLE && BT_BLUEDROID_ENABLED

Default value:
• 40 if CONFIG_BT_GATTC_ENABLE && BT_BLUEDROID_ENABLED

CONFIG_BT_GATTC_CACHE_NVS_FLASH

Save gattc cache data to nvs flash

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

This select can save gattc cache data to nvs flash

Default value:
• No (disabled) if CONFIG_BT_GATTC_ENABLE && BT_BLUEDROID_ENABLED

CONFIG_BT_GATTC_CONNECT_RETRY_COUNT

The number of attempts to reconnect if the connection establishment failed

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

The number of attempts to reconnect if the connection establishment failed

Range:
• from 0 to 7 if CONFIG_BT_GATTC_ENABLE && BT_BLUEDROID_ENABLED

Default value:
• 3 if CONFIG_BT_GATTC_ENABLE && BT_BLUEDROID_ENABLED

CONFIG_BT_BLE_SMP_ENABLE

Include BLE security module(SMP)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be close when the app not used the ble security connect.

Default value:
• Yes (enabled) if CONFIG_BT_BLE_ENABLED && BT_BLUEDROID_ENABLED
**CONFIG_BT_SMP_SLAVE_CON_PARAMS_UPD_ENABLE**

Slave enable connection parameters update during pairing

*Found in: Component config > Bluetooth > Bluebird Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_BLE_SMP_ENABLE*

In order to reduce the pairing time, slave actively initiates connection parameters update during pairing.

**Default value:**
- No (disabled) if `CONFIG_BT_BLE_SMP_ENABLE` && `BT_BLUE_DROID_ENABLED`

**CONFIG_BT_STACK_NO_LOG**

Disable BT debug logs (minimize bin size)

*Found in: Component config > Bluetooth > Bluebird Options*

This select can save the rodata code size

**Default value:**
- No (disabled) if `BT_BLUE_DROID_ENABLED` && `BT_BLUE_DROID_ENABLED`

**BT_DEBUG_LOG_LEVEL**

Contains:

- `CONFIG_BT_LOG_A2D_TRACE_LEVEL`
- `CONFIG_BT_LOG_APPL_TRACE_LEVEL`
- `CONFIG_BT_LOG_AVCT_TRACE_LEVEL`
- `CONFIG_BT_LOG_AVDT_TRACE_LEVEL`
- `CONFIG_BT_LOG_AVRC_TRACE_LEVEL`
- `CONFIG_BT_LOG_BLUFI_TRACE_LEVEL`
- `CONFIG_BT_LOG_BNEP_TRACE_LEVEL`
- `CONFIG_BT_LOG_BTC_TRACE_LEVEL`
- `CONFIG_BT_LOG_BTIF_TRACE_LEVEL`
- `CONFIG_BT_LOG_BTM_TRACE_LEVEL`
- `CONFIG_BT_LOG_GAP_TRACE_LEVEL`
- `CONFIG_BT_LOG_GATT_TRACE_LEVEL`
- `CONFIG_BT_LOG_HCI_TRACE_LEVEL`
- `CONFIG_BT_LOG_HID_TRACE_LEVEL`
- `CONFIG_BT_LOG_L2CAP_TRACE_LEVEL`
- `CONFIG_BT_LOG_MCA_TRACE_LEVEL`
- `CONFIG_BT_LOG_OSI_TRACE_LEVEL`
- `CONFIG_BT_LOG_PATCH_TRACE_LEVEL`
- `CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL`
- `CONFIG_BT_LOG_SDP_TRACE_LEVEL`
- `CONFIG_BT_LOG_SMP_TRACE_LEVEL`
- `CONFIG_BT_LOG_HCI_TRACE_LEVEL`

*Define BT trace level for HCI layer*

**Available options:**

- NONE (`BT_LOG_HCI_TRACE_LEVEL_NONE`)
- ERROR (`BT_LOG_HCI_TRACE_LEVEL_ERROR`)
- WARNING (`BT_LOG_HCI_TRACE_LEVEL_WARNING`)
- API (`BT_LOG_HCI_TRACE_LEVEL_API`)
- EVENT (`BT_LOG_HCI_TRACE_LEVEL_EVENT`)
- DEBUG (`BT_LOG_HCI_TRACE_LEVEL_DEBUG`)

---

**CONFIG_BT_LOG_HCI_TRACE_LEVEL**

HCI layer

*Found in: Component config > Bluetooth > Bluebird Options > BT_DEBUG_LOG_LEVEL*

Define BT trace level for HCI layer

**Available options:**

- NONE (`BT_LOG_HCI_TRACE_LEVEL_NONE`)
- ERROR (`BT_LOG_HCI_TRACE_LEVEL_ERROR`)
- WARNING (`BT_LOG_HCI_TRACE_LEVEL_WARNING`)
- API (`BT_LOG_HCI_TRACE_LEVEL_API`)
- EVENT (`BT_LOG_HCI_TRACE_LEVEL_EVENT`)
- DEBUG (`BT_LOG_HCI_TRACE_LEVEL_DEBUG`)

---

Espressif Systems 1367 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
Chapter 2. API

• VERBOSE (BT_LOG_HCI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTM_TRACE_LEVEL

BTM layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BTM layer

Available options:
• NONE (BT_LOG_BTM_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_BTM_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_BTM_TRACE_LEVEL_WARNING)
• API (BT_LOG_BTM_TRACE_LEVEL_API)
• EVENT (BT_LOG_BTM_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_BTM_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_BTM_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_L2CAP_TRACE_LEVEL

L2CAP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for L2CAP layer

Available options:
• NONE (BT_LOG_L2CAP_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_L2CAP_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_L2CAP_TRACE_LEVEL_WARNING)
• API (BT_LOG_L2CAP_TRACE_LEVEL_API)
• EVENT (BT_LOG_L2CAP_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_L2CAP_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_L2CAP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL

RFCOMM layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for RFCOMM layer

Available options:
• NONE (BT_LOG_RFCOMM_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_RFCOMM_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_RFCOMM_TRACE_LEVEL_WARNING)
• API (BT_LOG_RFCOMM_TRACE_LEVEL_API)
• EVENT (BT_LOG_RFCOMM_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_RFCOMM_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_RFCOMM_TRACE_LEVELVERBOSE)

CONFIG_BT_LOG_SDP_TRACE_LEVEL

SDP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for SDP layer

Available options:
• NONE (BT_LOG_SDP_TRACE_LEVEL_NONE)
Chapter 2. API

• ERROR (BT_LOG_SDP_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_SDP_TRACE_LEVEL_WARNING)
• API (BT_LOG_SDP_TRACE_LEVEL_API)
• EVENT (BT_LOG_SDP_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_SDP_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_SDP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_GAP_TRACE_LEVEL

GAP layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for GAP layer

Available options:
• NONE (BT_LOG_GAP_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_GAP_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_GAP_TRACE_LEVEL_WARNING)
• API (BT_LOG_GAP_TRACE_LEVEL_API)
• EVENT (BT_LOG_GAP_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_GAP_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_GAP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BNEP_TRACE_LEVEL

BNEP layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for BNEP layer

Available options:
• NONE (BT_LOG_BNEP_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_BNEP_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_BNEP_TRACE_LEVEL_WARNING)
• API (BT_LOG_BNEP_TRACE_LEVEL_API)
• EVENT (BT_LOG_BNEP_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_BNEP_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_BNEP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_PAN_TRACE_LEVEL

PAN layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for PAN layer

Available options:
• NONE (BT_LOG_PAN_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_PAN_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_PAN_TRACE_LEVEL_WARNING)
• API (BT_LOG_PAN_TRACE_LEVEL_API)
• EVENT (BT_LOG_PAN_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_PAN_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_PAN_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_A2D_TRACE_LEVEL
A2D layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for A2D layer

**Available options:**
- NONE (BT_LOG_A2D_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_A2D_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_A2D_TRACE_LEVEL_WARNING)
- API (BT_LOG_A2D_TRACE_LEVEL_API)
- EVENT (BT_LOG_A2D_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_A2D_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_A2D_TRACE_LEVEL_VERBOSE)

**CONFIG_BT_LOG_AVDT_TRACE_LEVEL**

AVDT layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for AVDT layer

**Available options:**
- NONE (BT_LOG_AVDT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVDT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVDT_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVDT_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVDT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVDT_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_AVDT_TRACE_LEVEL_VERBOSE)

**CONFIG_BT_LOG_AVCT_TRACE_LEVEL**

AVCT layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for AVCT layer

**Available options:**
- NONE (BT_LOG_AVCT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVCT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVCT_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVCT_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVCT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVCT_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_AVCT_TRACE_LEVEL_VERBOSE)

**CONFIG_BT_LOG_AVRC_TRACE_LEVEL**

AVRC layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for AVRC layer

**Available options:**
- NONE (BT_LOG_AVRC_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVRC_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVRC_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVRC_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVRC_TRACE_LEVEL_EVENT)
Chapter 2. API

- DEBUG (BT_LOG_AVRC_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_AVRC_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_MCA_TRACE_LEVEL

MCA layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for MCA layer

**Available options:**
- NONE (BT_LOG_MCA_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_MCA_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_MCA_TRACE_LEVEL_WARNING)
- API (BT_LOG_MCA_TRACE_LEVEL_API)
- EVENT (BT_LOG_MCA_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_MCA_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_MCA_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_HID_TRACE_LEVEL

HID layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for HID layer

**Available options:**
- NONE (BT_LOG_HID_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_HID_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_HID_TRACE_LEVEL_WARNING)
- API (BT_LOG_HID_TRACE_LEVEL_API)
- EVENT (BT_LOG_HID_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_HID_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_HID_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_APPL_TRACE_LEVEL

APPL layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for APPL layer

**Available options:**
- NONE (BT_LOG_APPL_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_APPL_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_APPL_TRACE_LEVEL_WARNING)
- API (BT_LOG_APPL_TRACE_LEVEL_API)
- EVENT (BT_LOG_APPL_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_APPL_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_APPL_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_GATT_TRACE_LEVEL

GATT layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for GATT layer

**Available options:**
• NONE (BT_LOG_GATT_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_GATT_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_GATT_TRACE_LEVEL_WARNING)
• API (BT_LOG_GATT_TRACE_LEVEL_API)
• EVENT (BT_LOG_GATT_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_GATT_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_GATT_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_SMP_TRACE_LEVEL

SMP layer

_Found in: Component config > Bluetooth > Bluez > BluezDroid Options > BT DEBUG LOG LEVEL_

Define BT trace level for SMP layer

_Available options:_
• NONE (BT_LOG_SMP_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_SMP_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_SMP_TRACE_LEVEL_WARNING)
• API (BT_LOG_SMP_TRACE_LEVEL_API)
• EVENT (BT_LOG_SMP_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_SMP_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_SMP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTIF_TRACE_LEVEL

BTIF layer

_Found in: Component config > Bluetooth > Bluez > BluezDroid Options > BT DEBUG LOG LEVEL_

Define BT trace level for BTIF layer

_Available options:_
• NONE (BT_LOG_BTIF_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_BTIF_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_BTIF_TRACE_LEVEL_WARNING)
• API (BT_LOG_BTIF_TRACE_LEVEL_API)
• EVENT (BT_LOG_BTIF_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_BTIF_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_BTIF_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTC_TRACE_LEVEL

BTC layer

_Found in: Component config > Bluetooth > Bluez > BluezDroid Options > BT DEBUG LOG LEVEL_

Define BT trace level for BTC layer

_Available options:_
• NONE (BT_LOG_BTC_TRACE_LEVEL_NONE)
• ERROR (BT_LOG_BTC_TRACE_LEVEL_ERROR)
• WARNING (BT_LOG_BTC_TRACE_LEVEL_WARNING)
• API (BT_LOG_BTC_TRACE_LEVEL_API)
• EVENT (BT_LOG_BTC_TRACE_LEVEL_EVENT)
• DEBUG (BT_LOG_BTC_TRACE_LEVEL_DEBUG)
• VERBOSE (BT_LOG_BTC_TRACE_LEVEL_VERBOSE)
CONFIG_BT_LOG_OSI_TRACE_LEVEL

OSI layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for OSI layer

*Available options:*
- NONE (BT_LOG_OSI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_OSI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_OSI_TRACE_LEVEL_WARNING)
- API (BT_LOG_OSI_TRACE_LEVEL_API)
- EVENT (BT_LOG_OSI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_OSI_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_OSI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BLUFI_TRACE_LEVEL

BLUFI layer

*Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL*

Define BT trace level for BLUFI layer

*Available options:*
- NONE (BT_LOG_BLUFI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BLUFI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BLUFI_TRACE_LEVEL_WARNING)
- API (BT_LOG_BLUFI_TRACE_LEVEL_API)
- EVENT (BT_LOG_BLUFI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BLUFI_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_BLUFI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_ACL_CONNECTIONS

BT/BLE MAX ACL CONNECTIONS(1~7)

*Found in: Component config > Bluetooth > Bluedroid Options*

Maximum BT/BLE connection count

*Range:*
- from 1 to 7 if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

*Default value:*
- 4 if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_MULTI_CONNECTION_ENBALE

Enable BLE multi-connections

*Found in: Component config > Bluetooth > Bluedroid Options*

Enable this option if there are multiple connections

*Default value:*
- Yes (enabled) if BT_BLUEDROID_ENABLED && BT_BLUEDROID_ENABLED

CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST

BT/BLE will first malloc the memory from the PSRAM

*Found in: Component config > Bluetooth > Bluedroid Options*

This select can save the internal RAM if there have the PSRAM
**Default value:**
- No (disabled) if BT_BLUEDROID_ENABLED & & BT_BLUEDROID_ENABLED

**CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY**

Use dynamic memory allocation in BT/BLE stack

*Found in: Component config > Bluetooth > Bluez Options*

This select can make the allocation of memory will become more flexible

**Default value:**
- No (disabled) if BT_BLUEDROID_ENABLED & & BT_BLUEDROID_ENABLED

**CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK**

BLE queue congestion check

*Found in: Component config > Bluetooth > Bluez Options*

When scanning and scan duplicate is not enabled, if there are a lot of adv packets around or application layer handling adv packets is slow, it will cause the controller memory to run out. if enabled, adv packets will be lost when host queue is congested.

**Default value:**
- No (disabled) if BT_BLUEDROID_ENABLED & & BT_BLUEDROID_ENABLED

**CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN**

Report adv data and scan response individually when BLE active scan

*Found in: Component config > Bluetooth > Bluez Options*

Originally, when doing BLE active scan, Bluez will not report adv to application layer until receive scan response. This option is used to disable the behavior. When enable this option, Bluez will report adv data or scan response to application layer immediately.

* # Memory reserved at start of DRAM for Bluetooth stack

**Default value:**
- No (disabled) if BT_BLUEDROID_ENABLED & & CONFIG_BT_BLE_ENABLED & & BT_BLUEDROID_ENABLED

**CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT**

Timeout of BLE connection establishment

*Found in: Component config > Bluetooth > Bluez Options*

Bluetooth Connection establishment maximum time, if connection time exceeds this value, the connection establishment fails, ESP_GATTC_OPEN_EVT or ESP_GATTS_OPEN_EVT is triggered.

**Range:**
- from 1 to 60 if BT_BLUEDROID_ENABLED & & BT_BLUEDROID_ENABLED

**Default value:**
- 30 if BT_BLUEDROID_ENABLED & & BT_BLUEDROID_ENABLED

**CONFIG_BT_MAX_DEVICE_NAME_LEN**

length of bluetooth device name

*Found in: Component config > Bluetooth > Bluez Options*
Bluetooth Device name length shall be no larger than 248 octets. If the broadcast data cannot contain the complete device name, then only the shortname will be displayed, the rest parts that can’t fit in will be truncated.

**Range:**
- from 32 to 248 if `BT_BLUEDROID_ENABLED` && `BT_BLUEDROID_ENABLED`

**Default value:**
- 32 if `BT_BLUEDROID_ENABLED` && `BT_BLUEDROID_ENABLED`

**CONFIG_BT_BLE_RPA_SUPPORTED**

Update RPA to Controller

*Found in: Component config > Bluetooth > Bluedroid Options*

This enables controller RPA list function. For ESP32, ESP32 only support network privacy mode. If this option is enabled, ESP32 will only accept advertising packets from peer devices that contain private address, HW will not receive the advertising packets contain identity address after IRK changed. If this option is disabled, address resolution will be performed in the host, so the functions that require controller to resolve address in the white list cannot be used. This option is disabled by default on ESP32, please enable or disable this option according to your own needs.

For ESP32C3, ESP32S3, ESP32H4 and ESP32C2, devices support network privacy mode and device privacy mode, users can switch the two modes according to their own needs. So this option is enabled by default.

**Default value:**
- No (disabled) if `BT_BLUEDROID_ENABLED` && `BT_BLUEDROID_ENABLED`
- Yes (enabled) if `BT_BLUEDROID_ENABLED` && `BT_BLUEDROID_ENABLED`

**CONFIG_BT_BLE_50_FEATURES_SUPPORTED**

Enable BLE 5.0 features

*Found in: Component config > Bluetooth > Bluedroid Options*

This enables BLE 5.0 features, this option only support esp32c3/esp32s3 chip

**Default value:**
- Yes (enabled) if `BT_BLUEDROID_ENABLED` && `SOC_ESP_NIMBLE_CONTROLLER` && `BT_BLUEDROID_ENABLED`

**CONFIG_BT_BLE_42_FEATURES_SUPPORTED**

Enable BLE 4.2 features

*Found in: Component config > Bluetooth > Bluedroid Options*

This enables BLE 4.2 features.

**Default value:**
- No (disabled) if `BT_BLUEDROID_ENABLED` && `SOC_ESP_NIMBLE_CONTROLLER` && `BT_BLUEDROID_ENABLED`

**NimBLE Options**

Contains:

- `CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME`
- `CONFIG_BT_NIMBLE_HS_STOP_TIMEOUT_MS`
- `CONFIG_BT_NIMBLE_WHITELIST_SIZE`
- `CONFIG_BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM`
- `CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT`
- `CONFIG_BT_NIMBLE_52_FEATURE_SUPPORT`
- `CONFIG_BT_NIMBLE_53_FEATURE_SUPPORT`
Chapter 2. API

- CONFIG_BT_NIMBLE_ROLE_BROADCASTER
- CONFIG_BT_NIMBLE_ROLE_CENTRAL
- CONFIG_BT_NIMBLE_MESH
- CONFIG_BT_NIMBLE_ROLE_OBSERVER
- CONFIG_BT_NIMBLE_ROLE_PERIPHERAL
- CONFIG_BT_NIMBLE_SECURITY_ENABLE
- CONFIG_BT_NIMBLE_BLUFI_ENABLE
- CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT
- CONFIG_BT_NIMBLE_USE_ESP_TIMER
- CONFIG_BT_NIMBLE_DEBUG
- CONFIG_BT_NIMBLE_HS_FLOW_CTRL
- CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE
- CONFIG_BT_NIMBLE_MAX_BONDS
- CONFIG_BT_NIMBLE_MAX_CCCDS
- CONFIG_BT_NIMBLE_MAX_CONNECTIONS
- CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM
- CONFIG_BT_NIMBLE_GATT_MAX_PROCS
- CONFIG_BT_NIMBLE_MEM_ALLOC_MODE
- Memory Settings
  - CONFIG_BT_NIMBLE_LOG_LEVEL
  - CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE
  - CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS
  - CONFIG_BT_NIMBLE_NVS_PERSIST
  - CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU
  - CONFIG_BT_NIMBLE_RPA_TIMEOUT
  - CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE
  - CONFIG_BT_NIMBLE_TEST_THROUGHPUT_TEST

CONFIG_BT_NIMBLE_MEM_ALLOC_MODE

Memory allocation strategy

Found in: Component config > Bluetooth > NimBLE Options

Allocation strategy for NimBLE host stack, essentially provides ability to allocate all required dynamic allocations from,

- Internal DRAM memory only
- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF
- Internal IRAM memory wherever applicable else internal DRAM

Available options:

- Internal memory (BT_NIMBLE_MEM_ALLOC_MODE_INTERNAL)
- External SPIRAM (BT_NIMBLE_MEM_ALLOC_MODE_EXTERNAL)
- Default alloc mode (BT_NIMBLE_MEM_ALLOC_MODE_DEFAULT)
- Internal IRAM (BT_NIMBLE_MEM_ALLOC_MODE_IRAM_8BIT)
  Allows to use IRAM memory region as 8bit accessible region.
  Every unaligned (8bit or 16bit) access will result in an exception and incur penalty of certain clock cycles per unaligned read/write.

CONFIG_BT_NIMBLE_LOG_LEVEL

NimBLE Host log verbosity

Found in: Component config > Bluetooth > NimBLE Options

Select NimBLE log level. Please make a note that the selected NimBLE log verbosity can not exceed the level set in “Component config -> Log output -> Default log verbosity”.
Available options:

- No logs (BT_NIMBLE_LOG_LEVEL_NONE)
- Error logs (BT_NIMBLE_LOG_LEVEL_ERROR)
- Warning logs (BT_NIMBLE_LOG_LEVEL_WARNING)
- Info logs (BT_NIMBLE_LOG_LEVEL_INFO)
- Debug logs (BT_NIMBLE_LOG_LEVEL_DEBUG)

**CONFIG_BT_NIMBLE_MAX_CONNECTIONS**

Maximum number of concurrent connections

*Found in: Component config > Bluetooth > NimBLE Options*

Defines maximum number of concurrent BLE connections. For ESP32, user is expected to configure BTDM_CTRL_BLE_MAX_CONN from controller menu along with this option. Similarly for ESP32-C3 or ESP32-S3, user is expected to configure BT_CTRL_BLE_MAX_ACT from controller menu.

*Range:*
- from 1 to 9 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_MAX_BONDS**

Maximum number of bonds to save across reboots

*Found in: Component config > Bluetooth > NimBLE Options*

Defines maximum number of bonds to save for peer security and our security

*Default value:*
- 3 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_MAX_CCCDS**

Maximum number of CCC descriptors to save across reboots

*Found in: Component config > Bluetooth > NimBLE Options*

Defines maximum number of CCC descriptors to save

*Default value:*
- 8 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM**

Maximum number of connection oriented channels

*Found in: Component config > Bluetooth > NimBLE Options*

Defines maximum number of BLE Connection Oriented Channels. When set to (0), BLE COC is not compiled in

*Range:*
- from 0 to 9 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

*Default value:*
- 0 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE**

The CPU core on which NimBLE host will run

*Found in: Component config > Bluetooth > NimBLE Options*

The CPU core on which NimBLE host will run. You can choose Core 0 or Core 1. Cannot specify no-affinity
Available options:

- Core 0 (PRO CPU) (BT_NIMBLE_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (BT_NIMBLE_PINNED_TO_CORE_1)

**CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE**

NimBLE Host task stack size

*Found in: Component config > Bluetooth > NimBLE Options*

This configures stack size of NimBLE host task

**Default value:**

- 5120 if `CONFIG_BLE_MESH` && `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`
- 4096 if `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_ROLE_CENTRAL**

Enable BLE Central role

*Found in: Component config > Bluetooth > NimBLE Options*

Enable central role

**Default value:**

- Yes (enabled) if `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_ROLE_PERIPHERAL**

Enable BLE Peripheral role

*Found in: Component config > Bluetooth > NimBLE Options*

Enable peripheral role

**Default value:**

- Yes (enabled) if `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_ROLE_BROADCASTER**

Enable BLE Broadcaster role

*Found in: Component config > Bluetooth > NimBLE Options*

Enable broadcaster role

**Default value:**

- Yes (enabled) if `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_ROLE_OBSERVER**

Enable BLE Observer role

*Found in: Component config > Bluetooth > NimBLE Options*

Enable observer role

**Default value:**

- Yes (enabled) if `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`
CONFIG_BT_NIMBLE_NVS_PERSIST
Persist the BLE Bonding keys in NVS

*Found in: Component config > Bluetooth > NimBLE Options*
Enable this flag to make bonding persistent across device reboots

**Default value:**
- No (disabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_SECURITY_ENABLE
Enable BLE SM feature

*Found in: Component config > Bluetooth > NimBLE Options*
Enable BLE sm feature

**Default value:**
- Yes (enabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Contains:
- **CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_ENCRYPTION**
- **CONFIG_BT_NIMBLE_SM_LEGACY**
- **CONFIG_BT_NIMBLE_SM_SC**

CONFIG_BT_NIMBLE_SM_LEGACY
Security manager legacy pairing

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE*
Enable security manager legacy pairing

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_SECURITY_ENABLE && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_SM_SC
Security manager secure connections (4.2)

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE*
Enable security manager secure connections

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_SECURITY_ENABLE && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_SM_SC_DEBUG_KEYS
Use predefined public-private key pair

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_SECURITY_ENABLE > CONFIG_BT_NIMBLE_SM_SC*
If this option is enabled, SM uses predefined DH key pair as described in Core Specification, Vol. 3, Part H, 2.3.5.6.1. This allows to decrypt air traffic easily and thus should only be used for debugging.

**Default value:**
• No (disabled) if `CONFIG_BT_NIMBLE_SECURITY_ENABLE` && `CONFIG_BT_NIMBLE_SM_SC` && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_ENCRYPTION**

Enable LE encryption

*Found in:* Component config > Bluetooth > NimBLE Options > `CONFIG_BT_NIMBLE_SECURITY_ENABLE`

Enable encryption connection

**Default value:**

• Yes (enabled) if `CONFIG_BT_NIMBLE_SECURITY_ENABLE` && BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_DEBUG**

Enable extra runtime asserts and host debugging

*Found in:* Component config > Bluetooth > NimBLE Options

This enables extra runtime asserts and host debugging

**Default value:**

• No (disabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME**

BLE GAP default device name

*Found in:* Component config > Bluetooth > NimBLE Options

The Device Name characteristic shall contain the name of the device as an UTF-8 string. This name can be changed by using API ble_svc_gap_device_name_set()

**Default value:**

• “nimble” if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN**

Maximum length of BLE device name in octets

*Found in:* Component config > Bluetooth > NimBLE Options

Device Name characteristic value shall be 0 to 248 octets in length

**Default value:**

• 31 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU**

Preferred MTU size in octets

*Found in:* Component config > Bluetooth > NimBLE Options

This is the default value of ATT MTU indicated by the device during an ATT MTU exchange. This value can be changed using API ble_att_set_preferred_mtu()

**Default value:**

• 256 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
**CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE**

External appearance of the device

*Found in: Component config > Bluetooth > NimBLE Options*

Standard BLE GAP Appearance value in HEX format e.g. 0x02C0

**Default value:**
- 0 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**Memory Settings**

Contains:

- `CONFIG_BT_NIMBLE_ACL_BUF_COUNT`
- `CONFIG_BT_NIMBLE_ACL_BUF_SIZE`
- `CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE`
- `CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT`
- `CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT`
- `CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT`
- `CONFIG_BT_NIMBLE_MSYS_1_BLOCK_SIZE`
- `CONFIG_BT_NIMBLE_MSYS_2_BLOCK_COUNT`
- `CONFIG_BT_NIMBLE_MSYS_2_BLOCK_SIZE`

**CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT**

MSYS_1 Block Count

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

MSYS is a system level mbuf registry. For prepare write & prepare responses MBUFs are allocated out of msys_1 pool. For NIMBLE_MESH enabled cases, this block count is increased by 8 than user defined count.

**Default value:**
- 24 if SOC_ESP_NIMBLE_CONTROLLER && BT_NIMBLE_ENABLED
- 12 if SOC_ESP_NIMBLE_CONTROLLER && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_MSYS_1_BLOCK_SIZE**

MSYS_1 Block Size

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

Dynamic memory size of block 1

**Default value:**
- 128 if SOC_ESP_NIMBLE_CONTROLLER && BT_NIMBLE_ENABLED
- 256 if SOC_ESP_NIMBLE_CONTROLLER && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_MSYS_2_BLOCK_COUNT**

MSYS_2 Block Count

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

Dynamic memory count

**Default value:**
- 24 if BT_NIMBLE_ENABLED
CONFIG_BT_NIMBLE_MSYS_2_BLOCK_SIZE
MSYS_2 Block Size

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

Dynamic memory size of block 2

**Default value:**
- 320 if BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_ACL_BUF_COUNT
ACL Buffer count

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

The number of ACL data buffers.

**Default value:**
- 24 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_ACL_BUF_SIZE
ACL Buffer size

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

This is the maximum size of the data portion of HCI ACL data packets. It does not include the HCI data header (of 4 bytes)

**Default value:**
- 255 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE
HCI Event Buffer size

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

This is the size of each HCI event buffer in bytes. In case of extended advertising, packets can be fragmented. 257 bytes is the maximum size of a packet.

**Default value:**
- 257 if CONFIG_BT_NIMBLE_EXT_ADV && BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
- 70 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT
High Priority HCI Event Buffer count

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

This is the high priority HCI events’ buffer size. High-priority event buffers are for everything except advertising reports. If there are no free high-priority event buffers then host will try to allocate a low-priority buffer instead.

**Default value:**
- 30 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT
Low Priority HCI Event Buffer count

*Found in: Component config > Bluetooth > NimBLE Options > Memory Settings*

This is the low priority HCI events’ buffer size. Low-priority event buffers are only used for advertising reports. If there are no free low-priority event buffers, then an incoming advertising report will get dropped.

**Default value:**
- 8 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_GATT_MAX_PROCS
Maximum number of GATT client procedures

*Found in: Component config > Bluetooth > NimBLE Options*

Maximum number of GATT client procedures that can be executed.

**Default value:**
- 4 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HS_FLOW_CTRL
Enable Host Flow control

*Found in: Component config > Bluetooth > NimBLE Options*

Enable Host Flow control

**Default value:**
- Yes (enabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
- No (disabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_ITVL
Host Flow control interval

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL*

Host flow control interval in msecs

**Default value:**
- 1000 if CONFIG_BT_NIMBLE_HS_FLOW_CTRL && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_THRESH
Host Flow control threshold

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL*

Host flow control threshold, if the number of free buffers are at or below this threshold, send an immediate number-of-completed-packets event

**Default value:**
- 2 if CONFIG_BT_NIMBLE_HS_FLOW_CTRL && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT
Host Flow control on disconnect

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL*

Enable this option to send number-of-completed-packets event to controller after disconnection
Default value:
  • Yes (enabled) if CONFIG_BT_NIMBLE_HS_FLOW_CTRL && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_RPA_TIMEOUT

RPA timeout in seconds

*Found in: Component config > Bluetooth > NimBLE Options*

Time interval between RPA address change. This is applicable in case of Host based RPA

*Range:*
  • from 1 to 41400 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

*Default value:*
  • 900 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_MESH

Enable BLE mesh functionality

*Found in: Component config > Bluetooth > NimBLE Options*

Enable BLE Mesh functionality

*Default value:*
  • No (disabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Contains:
  • CONFIG_BT_NIMBLE_MESH_PROVISIONER
  • CONFIG_BT_NIMBLE_MESH_PROV
  • CONFIG_BT_NIMBLE_MESH_GATT_PROXY
  • CONFIG_BT_NIMBLE_MESH_FRIEND
  • CONFIG_BT_NIMBLE_MESH_LOW_POWER
  • CONFIG_BT_NIMBLE_MESH_PROXY
  • CONFIG_BT_NIMBLE_MESH_RELAY
  • CONFIG_BT_NIMBLE_MESH_DEVICE_NAME
  • CONFIG_BT_NIMBLE_MESH_NODE_COUNT

CONFIG_BT_NIMBLE_MESH_PROXY

Enable mesh proxy functionality

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH*

Enable proxy. This is automatically set whenever NIMBLE_MESH_PB_GATT or NIM-BLE_MESH_GATT_PROXY is set

*Default value:*
  • No (disabled) if CONFIG_BT_NIMBLE_MESH && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_MESH_PROV

Enable BLE mesh provisioning

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH*

Enable mesh provisioning

*Default value:*
  • Yes (enabled) if CONFIG_BT_NIMBLE_MESH && BT_NIMBLE_ENABLED
CONFIG_BT_NIMBLE_MESH_PB_ADV
Enable mesh provisioning over advertising bearer

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH > CONFIG_BT_NIMBLE_MESH_PROV

Enable this option to allow the device to be provisioned over the advertising bearer

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_MESH_PROV && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_MESH_PB_GATT
Enable mesh provisioning over GATT bearer

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH > CONFIG_BT_NIMBLE_MESH_PROV

Enable this option to allow the device to be provisioned over the GATT bearer

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_MESH_PROV && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_MESH_GATT_PROXY
Enable GATT Proxy functionality

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

This option enables support for the Mesh GATT Proxy Service, i.e. the ability to act as a proxy between a Mesh GATT Client and a Mesh network

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_MESH && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_MESH_RELAY
Enable mesh relay functionality

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Support for acting as a Mesh Relay Node

**Default value:**
- No (disabled) if CONFIG_BT_NIMBLE_MESH && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_MESH_LOW_POWER
Enable mesh low power mode

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable this option to be able to act as a Low Power Node

**Default value:**
- No (disabled) if CONFIG_BT_NIMBLE_MESH && BT_NIMBLE_ENABLED

CONFIG_BT_NIMBLE_MESH_FRIEND
Enable mesh friend functionality

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable this option to be able to act as a Friend Node

**Default value:**
• No (disabled) if `CONFIG_BT_NIMBLE_MESH` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_MESH_DEVICE_NAME**

Set mesh device name

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH*

This value defines Bluetooth Mesh device/node name

**Default value:**

• “nimble-mesh-node” if `CONFIG_BT_NIMBLE_MESH` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_MESH_NODE_COUNT**

Set mesh node count

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH*

Defines mesh node count.

**Default value:**

• 1 if `CONFIG_BT_NIMBLE_MESH` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_MESH_PROVISIONER**

Enable BLE mesh provisioner

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH*

Enable mesh provisioner.

**Default value:**

• 0 if `CONFIG_BT_NIMBLE_MESH` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS**

Override TinyCrypt with mbedTLS for crypto computations

*Found in: Component config > Bluetooth > NimBLE Options*

Enable this option to choose mbedTLS instead of TinyCrypt for crypto computations.

**Default value:**

• Yes (enabled) if `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_HS_STOP_TIMEOUT_MS**

BLE host stop timeout in msec

*Found in: Component config > Bluetooth > NimBLE Options*

BLE Host stop procedure timeout in milliseconds.

**Default value:**

• 2000 if `BT_NIMBLE_ENABLED` && `BT_NIMBLE_ENABLED`

**CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT**

Enable connection reattempts on connection establishment error

*Found in: Component config > Bluetooth > NimBLE Options*

Enable to make the NimBLE host to reattempt GAP connection on connection establishment failure.

**Default value:**
• No (disabled) if BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_MAX_CONN_REATTEMPT**

Maximum number connection reattempts

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT

Defines maximum number of connection reattempts.

**Range:**
- from 1 to 7 if BT_NIMBLE_ENABLED && CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT && BT_NIMBLE_ENABLED

**Default value:**
- 3 if BT_NIMBLE_ENABLED && CONFIG_BT_NIMBLE_ENABLE_CONN_REATTEMPT && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT**

Enable BLE 5 feature

*Found in:* Component config > Bluetooth > NimBLE Options

Enable BLE 5 feature

**Default value:**
- Yes (enabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

Contains:
- CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_2M_PHY
- CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_CODED_PHY
- CONFIG_BT_NIMBLE_EXT_ADV
- CONFIG_BT_NIMBLE_MAX_PERIODIC_SYNCS

**CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_2M_PHY**

Enable 2M Phy

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable 2M-PHY

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_LL_CFG_FEAT_LE_CODED_PHY**

Enable coded Phy

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable coded-PHY

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT && BT_NIMBLE_ENABLED
**CONFIG_BT_NIMBLE_EXT_ADV**

Enable extended advertising

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Enable this option to do extended advertising. Extended advertising will be supported from BLE 5.0 onwards.

**Default value:**
- No (disabled) if CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_MAX_EXT_ADV_INSTANCES**

Maximum number of extended advertising instances.

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV

Change this option to set maximum number of extended advertising instances. Minimum there is always one instance of advertising. Enter how many more advertising instances you want.

**Range:**
- from 0 to 4 if CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && BT_NIMBLE_ENABLED

**Default value:**
- 1 if CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && BT_NIMBLE_ENABLED
- 0 if CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_EXT_ADV_MAX_SIZE**

Maximum length of the advertising data.

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV

Defines the length of the extended adv data. The value should not exceed 1650.

**Range:**
- from 0 to 1650 if CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && && BT_NIMBLE_ENABLED

**Default value:**
- 1650 if CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && && BT_NIMBLE_ENABLED
- 0 if CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV**

Enable periodic advertisement.

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV

Enable this option to start periodic advertisement.

**Default value:**
- Yes (enabled) if CONFIG_BT_NIMBLE_EXT_ADV && CONFIG_BT_NIMBLE_EXT_ADV && && BT_NIMBLE_ENABLED
CONFIG_BT_NIMBLE_PERIODIC_ADV_SYNC_TRANSFER

Enable Transer Sync Events

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT > CONFIG_BT_NIMBLE_EXT_ADV > CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV

This enables controller transfer periodic sync events to host

*Default value:*

- Yes (enabled) if `CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV` && `CONFIG_BT_NIMBLE_EXT_ADV` && `BT_NIMBLE_ENABLED`

CONFIG_BT_NIMBLE_MAX_PERIODIC_SYNCS

Maximum number of periodic advertising syncs

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT

Set this option to set the upper limit for number of periodic sync connections. This should be less than maximum connections allowed by controller.

*Range:*

- from 0 to 8 if `CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT` && `BT_NIMBLE_ENABLED`

*Default value:*

- 1 if `CONFIG_BT_NIMBLE_ENABLE_PERIODIC_ADV` && `CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT` && `BT_NIMBLE_ENABLED`
- 0 if `CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT` && `BT_NIMBLE_ENABLED`

CONFIG_BT_NIMBLE_53_FEATURE_SUPPORT

Enable BLE 5.3 feature

*Found in:* Component config > Bluetooth > NimBLE Options

Enable BLE 5.3 feature

Contains:

- `CONFIG_BT_NIMBLE_SUBRATE`

CONFIG_BT_NIMBLE_SUBRATE

Connection Subrate

*Found in:* Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_53_FEATURE_SUPPORT

Enable support for Connection Subrate

CONFIG_BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM

Coexistence: limit on MAX Tx/Rx time for coded-PHY connection

*Found in:* Component config > Bluetooth > NimBLE Options

When using PHY-Coded in BLE connection, limitation on max tx/rx time can be applied to better avoid dramatic performance deterioration of Wi-Fi.

*Available options:*

- Force Enable (BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM_EN)
- Always enable the limitation on max tx/rx time for Coded-PHY connection
• Force Disable (BT_NIMBLE_COEX_PHY_CODED_TX_RX_TLIM_DIS)
  Disable the limitation on max tx/rx time for Coded-PHY connection

**CONFIG_BT_NIMBLE_52_FEATURE_SUPPORT**

Enable BLE 5.2 Feature

*Found in: Component config > Bluetooth > NimBLE Options*

Enable this option to select 5.2 features

Contains:

• **CONFIG_BT_NIMBLE_BLE_POWER_CONTROL**

**CONFIG_BT_NIMBLE_BLE_POWER_CONTROL**

Enable support for BLE Power Control

*Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_52_FEATURE_SUPPORT*

Set this option to enable the Power Control feature

**CONFIG_BT_NIMBLE_WHITELIST_SIZE**

BLE white list size

*Found in: Component config > Bluetooth > NimBLE Options*

BLE list size

*Range:*
  • from 1 to 15 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

*Default value:*
  • 12 if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_TEST_THROUGHPUT_TEST**

Throughput Test Mode enable

*Found in: Component config > Bluetooth > NimBLE Options*

Enable the throughput test mode

*Default value:*
  • No (disabled) if BT_NIMBLE_ENABLED

**CONFIG_BT_NIMBLE_BLUFI_ENABLE**

Enable blufi functionality

*Found in: Component config > Bluetooth > NimBLE Options*

Set this option to enable blufi functionality.

*Default value:*
  • No (disabled) if BT_NIMBLE_ENABLED && BT_NIMBLE_ENABLED
**CONFIG_BT_NIMBLE_USE_ESP_TIMER**

Enable Esp Timer for Nimble

*Found in: Component config > Bluetooth > NimBLE Options*

Set this option to use Esp Timer which has higher priority timer instead of FreeRTOS timer

**Default value:**
- Yes (enabled) if BT_NIMBLE_ENABLED

**Controller Options**

Contains:

- CONFIG_BTDM_CTRL_AUTO_LATENCY
- CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP
- CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED
- CONFIG_BTDM_CTRL_BLE_MAX_CONN
- CONFIG_BTDM_BLE_SCAN_DUPL
- CONFIG_BTDM_BLE_SLEEP_CLOCK_ACCURACY
- CONFIG_BTDM_CTRL_MODE
- CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN
- CONFIG_BTDM_CTRL_BR_EDR_MAX_SCO_DATA_PATH
- CONFIG_BTDM_CTRL_BR_EDR_MAX_SYNC_CONN
- CONFIG_BTDM_CTRL_HCI_MODE_CHOICE
- HCI UART(H4) Options
- CONFIG_BTDM_CTRL_HLI
- CONFIG_BTDM_CTRL_LEGACY_AUTH_VENDOR_EVT
- MODEM SLEEP Options
- CONFIG_BTDM_CTRL_PCM_ROLE_EDGE_CONFIG
- CONFIG_BTDM_CTRL_PINNED_TO_CORE_CHOICE

**CONFIG_BTDM_CTRL_MODE**

Bluetooth controller mode (BR/EDR/BLE/DUALMODE)

*Found in: Component config > Bluetooth > Controller Options*

Specify the bluetooth controller mode (BR/EDR, BLE or dual mode).

**Available options:**
- BLE Only (BTDM_CTRL_MODE_BLE_ONLY)
- BR/EDR Only (BTDM_CTRL_MODE_BR_EDR_ONLY)
- Bluetooth Dual Mode (BTDM_CTRL_MODE_BTDM)

**CONFIG_BTDM_CTRL_BLE_MAX_CONN**

BLE Max Connections

*Found in: Component config > Bluetooth > Controller Options*

BLE maximum connections of bluetooth controller. Each connection uses 1KB static DRAM whenever the BT controller is enabled.

**Range:**
- from 1 to 9 if (BTDM_CTRL_MODE_BLE_ONLY || BTDM_CTRL_MODE_BTDM) && BT_CONTROLLER_ENABLED

**Default value:**
- 3 if (BTDM_CTRL_MODE_BLE_ONLY || BTDM_CTRL_MODE_BTDM) && BT_CONTROLLER_ENABLED
**CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN**

BR/EDR ACL Max Connections

*Found in: Component config > Bluetooth > Controller Options*

BR/EDR ACL maximum connections of bluetooth controller. Each connection uses 1.2 KB DRAM whenever the BT controller is enabled.

**Range:**
- from 1 to 7 if (BTDM_CTRL_MODE_BR_EDR_ONLY || BTDM_CTRL_MODE_BTDM) && BT_CONTROLLER_ENABLED

**Default value:**
- 2 if (BTDM_CTRL_MODE_BR_EDR_ONLY || BTDM_CTRL_MODE_BTDM) && BT_CONTROLLER_ENABLED

**CONFIG_BTDM_CTRL_BR_EDR_MAX_SYNC_CONN**

BR/EDR Synchronize maximum connections of bluetooth controller. Each connection uses 2 KB DRAM whenever the BT controller is enabled.

**Range:**
- from 0 to 3 if (BTDM_CTRL_MODE_BR_EDR_ONLY || BTDM_CTRL_MODE_BTDM) && BT_CONTROLLER_ENABLED

**Default value:**
- 0 if (BTDM_CTRL_MODE_BR_EDR_ONLY || BTDM_CTRL_MODE_BTDM) && BT_CONTROLLER_ENABLED

**CONFIG_BTDM_CTRL_BR_EDR_SCO_DATA_PATH**

BR/EDR Sync(SCO/eSCO) default data path

*Found in: Component config > Bluetooth > Controller Options*

SCO data path, i.e. HCI or PCM. SCO data can be sent/received through HCI synchronous packets, or the data can be routed to on-chip PCM module on ESP32. PCM input/output signals can be “matrixed” to GPIOs. The default data path can also be set using API “esp_bredr_sco_datapath_set”

**Available options:**
- HCI (BTDM_CTRL_BR_EDR_SCO_DATA_PATH_HCI)
- PCM (BTDM_CTRL_BR_EDR_SCO_DATA_PATH_PCM)

**CONFIG_BTDM_CTRL_PCM_ROLE_EDGE_CONFIG**

PCM Signal Config (Role and Polar)

*Found in: Component config > Bluetooth > Controller Options*

**Default value:**
- Yes (enabled) if BTDM_CTRL_BR_EDR_SCO_DATA_PATH_PCM && BT_CONTROLLER_ENABLED

**Contains:**
- CONFIG_BTDM_CTRL_PCM_POLAR
- CONFIG_BTDM_CTRL_PCM_ROLE
Chapter 2. API

**CONFIG_BTDM_CTRL_PCM_ROLE**

PCM Role

*Found in: Component config > Bluetooth > Controller Options > CONFIG_BTDM_CTRL_PCM_ROLE_EDGE_CONFIG*

PCM role can be configured as PCM master or PCM slave

**Available options:**

- PCM Master (BTDM_CTRL_PCM_ROLE_MASTER)
- PCM Slave (BTDM_CTRL_PCM_ROLE_SLAVE)

**CONFIG_BTDM_CTRL_PCM_POLAR**

PCM Polar

*Found in: Component config > Bluetooth > Controller Options > CONFIG_BTDM_CTRL_PCM_ROLE_EDGE_CONFIG*

PCM polarity can be configured as Falling Edge or Rising Edge

**Available options:**

- Falling Edge (BTDM_CTRL_PCM_POLAR_FALLING_EDGE)
- Rising Edge (BTDM_CTRL_PCM_POLAR_RISING_EDGE)

**CONFIG_BTDM_CTRL_AUTO_LATENCY**

Auto latency

*Found in: Component config > Bluetooth > Controller Options*

BLE auto latency, used to enhance classic BT performance while classic BT and BLE are enabled at the same time.

**Default value:**

- No (disabled) if BTDM_CTRL_MODE_BTDM && BT_CONTROLLER_ENABLED

**CONFIG_BTDM_CTRL_LEGACY_AUTH_VENDOR_EVT**

Legacy Authentication Vendor Specific Event Enable

*Found in: Component config > Bluetooth > Controller Options*

To protect from BIAS attack during Legacy authentication, Legacy authentication Vendor specific event should be enabled

**Default value:**

- Yes (enabled) if (BTDM_CTRL_MODE_BR_EDR_ONLY || BTDM_CTRL_MODE_BTDM) && BT_CONTROLLER_ENABLED

**CONFIG_BTDM_CTRL_PINNED_TO_CORE_CHOICE**

The cpu core which bluetooth controller run

*Found in: Component config > Bluetooth > Controller Options*

Specify the cpu core to run bluetooth controller. Can not specify no-affinity.

**Available options:**

- Core 0 (PRO CPU) (BTDM_CTRL_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (BTDM_CTRL_PINNED_TO_CORE_1)
Chapter 2. API Reference

**CONFIG_BTDM_CTRL_HCI_MODE_CHOICE**

HCI mode

*Found in: Component config > Bluetooth > Controller Options*

Specify HCI mode as VHCI or UART(H4)

**Available options:**
- VHCI (BTDM_CTRL_HCI_MODE_VHCI)
  - Normal option. Mostly, choose this VHCI when Bluetooth host run on ESP32, too.
- UART(H4) (BTDM_CTRL_HCI_MODE_UART_H4)
  - If use external Bluetooth host which run on other hardware and use UART as the HCI interface, choose this option.

**HCI UART(H4) Options** Contains:
- `CONFIG_BTDM_CTRL_HCI_UART_BAUDRATE`
- `CONFIG_BTDM_CTRL_HCI_UART_NO`

**CONFIG_BTDM_CTRL_HCI_UART_NO**

UART Number for HCI

*Found in: Component config > Bluetooth > Controller Options > HCI UART(H4) Options*

Uart number for HCI. The available uart is UART1 and UART2.

**Range:**
- from 1 to 2 if BTDM_CTRL_HCI_MODE_UART_H4 && BT_CONTROLLER_ENABLED

**Default value:**
- 1 if BTDM_CTRL_HCI_MODE_UART_H4 && BT_CONTROLLER_ENABLED

**CONFIG_BTDM_CTRL_HCI_UART_BAUDRATE**

UART Baudrate for HCI

*Found in: Component config > Bluetooth > Controller Options > HCI UART(H4) Options*

UART Baudrate for HCI. Please use standard baudrate.

**Range:**
- from 115200 to 921600 if BTDM_CTRL_HCI_MODE_UART_H4 && BT_CONTROLLER_ENABLED

**Default value:**
- 921600 if BTDM_CTRL_HCI_MODE_UART_H4 && BT_CONTROLLER_ENABLED

**MODEM SLEEP Options** Contains:
- `CONFIG_BTDM_CTRL_LOW_POWER_CLOCK`
- `CONFIG_BTDM_CTRL_MODEM_SLEEP`

**CONFIG_BTDM_CTRL_MODEM_SLEEP**

Bluetooth modem sleep

*Found in: Component config > Bluetooth > Controller Options > MODEM SLEEP Options*

Enable/disable Bluetooth controller low power mode.

**Default value:**
- Yes (enabled) if BT_CONTROLLER_ENABLED
**CONFIG_BTDM_CTRL_MODEM_SLEEP_MODE**

Bluetooth Modem sleep mode

*Found in: Component config > Bluetooth > Controller Options > MODEM SLEEP Options > CONFIG_BTDM_CTRL_MODEM_SLEEP*

To select which strategy to use for modem sleep

**Available options:**
- **ORIG Mode (sleep with low power clock)** (BTDM_CTRL_MODEM_SLEEP_MODE_ORIG)
  ORIG mode is a Bluetooth sleep mode that can be used for dual mode controller. In this mode, Bluetooth controller sleeps between BR/EDR frames and BLE events. A low power clock is used to maintain Bluetooth reference clock.
- **EVED Mode (For internal test only)** (BTDM_CTRL_MODEM_SLEEP_MODE_EVED)
  EVED mode is for BLE only and is only for internal test. Do not use it for production. This mode is not compatible with DFS nor light sleep.

**CONFIG_BTDM_CTRL_LOW_POWER_CLOCK**

Bluetooth low power clock

*Found in: Component config > Bluetooth > Controller Options > MODEM SLEEP Options*

Select the low power clock source for Bluetooth controller. Bluetooth low power clock is the clock source to maintain time in sleep mode.

- “Main crystal” option provides good accuracy and can support Dynamic Frequency Scaling to be used with Bluetooth modem sleep. Light sleep is not supported.
- “External 32kHz crystal” option allows user to use a 32.768kHz crystal as Bluetooth low power clock. This option is allowed as long as External 32kHz crystal is configured as the system RTC clock source. This option provides good accuracy and supports Bluetooth modem sleep to be used alongside Dynamic Frequency Scaling or light sleep.

**Available options:**
- **Main crystal (BTDM_CTRL_LPCLK_SEL_MAIN_XTAL)**
  Main crystal can be used as low power clock for Bluetooth modem sleep. If this option is selected, Bluetooth modem sleep can work under Dynamic Frequency Scaling (DFS) enabled, but cannot work when light sleep is enabled. Main crystal has a good performance in accuracy as the Bluetooth low power clock source.
- **External 32kHz crystal (BTDM_CTRL_LPCLK_SEL_EXT_32K_XTAL)**
  External 32kHz crystal has a nominal frequency of 32.768kHz and provides good frequency stability. If used as Bluetooth low power clock, External 32kHz can support Bluetooth modem sleep to be used with both DFS and light sleep.

**CONFIG_BTDM_BLE_SLEEP_CLOCK_ACCURACY**

BLE Sleep Clock Accuracy

*Found in: Component config > Bluetooth > Controller Options*

BLE Sleep Clock Accuracy (SCA) for the local device is used to estimate window widening in BLE connection events. With a lower level of clock accuracy (e.g. 500ppm over 250ppm), the slave needs a larger RX window to synchronize with master in each anchor point, thus resulting in an increase of power consumption but a higher level of robustness in keeping connected. According to the requirements of Bluetooth Core specification 4.2, the worst-case accuracy of Classic Bluetooth low power oscialltor (LPO) is +/-250ppm in STANDBY and in low power modes such as sniff. For BLE the worst-case SCA is +/-500ppm.

- “151ppm to 250ppm” option is the default value for Bluetooth Dual mode
- “251ppm to 500ppm” option can be used in BLE only mode when using external 32kHz crystal as low power clock. This option is provided in case that BLE sleep clock has a lower level of accuracy, or other error sources contribute to the inaccurate timing during sleep.
Available options:
- 251ppm to 500ppm (BTDM_BLE_DEFAULT_SCA_500PPM)
- 151ppm to 250ppm (BTDM_BLE_DEFAULT_SCA_250PPM)

CONFIG_BTDM_BLE_SCAN_DUPL

BLE Scan Duplicate Options

*Found in: Component config > Bluetooth > Controller Options*

This select enables parameters setting of BLE scan duplicate.

**Default value:**
- Yes (enabled) if (BTDM_CTRL_MODE_BTDM || BTDM_CTRL_MODE_BLE_ONLY) && BT_CONTROLLER_ENABLED

CONFIG_BTDM_SCAN_DUPL_TYPE

Scan Duplicate Type

*Found in: Component config > Bluetooth > Controller Options > CONFIG_BTDM_BLE_SCAN_DUPL*

Scan duplicate have three ways. one is “Scan Duplicate By Device Address”，This way is to use advertiser address filtering. The adv packet of the same address is only allowed to be reported once. Another way is “Scan Duplicate By Device Address And Advertising Data” . This way is to use advertising data and device address filtering. All different adv packets with the same address are allowed to be reported. The last way is “Scan Duplicate By Advertising Data”. This way is to use advertising data filtering. All same advertising data only allow to be reported once even though they are from different devices.

**Available options:**
- Scan Duplicate By Device Address (BTDM_SCAN_DUPL_TYPE_DEVICE)
  This way is to use advertiser address filtering. The adv packet of the same address is only allowed to be reported once
- Scan Duplicate By Advertising Data (BTDM_SCAN_DUPL_TYPE_DATA)
  This way is to use advertising data filtering. All same advertising data only allow to be reported once even though they are from different devices.
- Scan Duplicate By Device Address And Advertising Data (BTDM_SCAN_DUPL_TYPE_DATA DEVICE)
  This way is to use advertising data and device address filtering. All different adv packets with the same address are allowed to be reported.

CONFIG_BTDM_SCAN_DUPL_CACHE_SIZE

Maximum number of devices in scan duplicate filter

*Found in: Component config > Bluetooth > Controller Options > CONFIG_BTDM_BLE_SCAN_DUPL*

Maximum number of devices which can be recorded in scan duplicate filter. When the maximum amount of device in the filter is reached, the cache will be refreshed.

**Range:**
- from 10 to 1000 if CONFIG_BTDM_BLE_SCAN_DUPL && BT_CONTROLLER_ENABLED

**Default value:**
- 100 if CONFIG_BTDM_BLE_SCAN_DUPL && BT_CONTROLLER_ENABLED

CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN

Special duplicate scan mechanism for BLE Mesh scan

*Found in: Component config > Bluetooth > Controller Options > CONFIG_BTDM_BLE_SCAN_DUPL*

This enables the BLE scan duplicate for special BLE Mesh scan.
Default value:
• No (disabled) if `CONFIG_BTDM_BLE_SCAN_DUPL` && `BT_CONTROLLER_ENABLED`

**CONFIG_BTDM_MESH_DUPL_SCAN_CACHE_SIZE**

Maximum number of Mesh adv packets in scan duplicate filter

*Found in: Component config > Bluetooth > Controller Options > CONFIG_BTDM_BLE_SCAN_DUPL > CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN*

Maximum number of adv packets which can be recorded in duplicate scan cache for BLE Mesh. When the maximum amount of device in the filter is reached, the cache will be refreshed.

**Range:**
• from 10 to 1000 if `CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN` && `BT_CONTROLLER_ENABLED`

**Default value:**
• 100 if `CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN` && `BT_CONTROLLER_ENABLED`

**CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED**

BLE full scan feature supported

*Found in: Component config > Bluetooth > Controller Options*

The full scan function is mainly used to provide BLE scan performance. This is required for scenes with high scan performance requirements, such as BLE Mesh scenes.

**Default value:**
• Yes (enabled) if (BTDM_CTRL_MODE_BLE_ONLY || BTDM_CTRL_MODE_BTDM) && `BT_CONTROLLER_ENABLED`

**CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP**

BLE adv report flow control supported

*Found in: Component config > Bluetooth > Controller Options*

The function is mainly used to enable flow control for advertising reports. When it is enabled, advertising reports will be discarded by the controller if the number of unprocessed advertising reports exceeds the size of BLE adv report flow control.

**Default value:**
• Yes (enabled) if (BTDM_CTRL_MODE_BTDM || BTDM_CTRL_MODE_BLE_ONLY) && `BT_CONTROLLER_ENABLED`

**CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM**

BLE adv report flow control number

*Found in: Component config > Bluetooth > Controller Options > CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP*

The number of unprocessed advertising report that Bluedroid can save. If you set `BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM` to a small value, this may cause adv packets lost. If you set `BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM` to a large value, Bluedroid may cache a lot of adv packets and this may cause system memory run out. For example, if you set it to 50, the maximum memory consumed by host is 35 * 50 bytes. Please set `BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM` according to your system free memory and handle adv packets as fast as possible, otherwise it will cause adv packets lost.

**Range:**
Chapter 2. API

**CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP**

- from 50 to 1000 if `CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP` && `BT_CONTROLLER_ENABLED`

**Default value:**
- 100 if `CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP` && `BT_CONTROLLER_ENABLED`

**CONFIG_BTDM_BLE_ADV_REPORT_DISCARD_THRESHOLD**

BLE adv lost event threshold value

*Found in:* Component config > Bluetooth > Controller Options > CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP

When adv report flow control is enabled, the ADV lost event will be generated when the number of ADV packets lost in the controller reaches this threshold. It is better to set a larger value. If you set `BTDM_BLE_ADV_REPORT_DISCARD_THRESHOLD` to a small value or printf every adv lost event, it may cause adv packets lost more.

**Range:**
- from 1 to 1000 if `CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP` && `BT_CONTROLLER_ENABLED`

**Default value:**
- 20 if `CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP` && `BT_CONTROLLER_ENABLED`

**CONFIG_BTDM_CTRL_HLI**

High level interrupt

*Found in:* Component config > Bluetooth > Controller Options

Using Level 4 interrupt for Bluetooth.

**Default value:**
- Yes (enabled) if `CONFIG_BT_ENABLED` && `BT_CONTROLLER_ENABLED`

**CONFIG_BLE_MESH**

ESP BLE Mesh Support

*Found in:* Component config

This option enables ESP BLE Mesh support. The specific features that are available may depend on other features that have been enabled in the stack, such as Bluetooth Support, Bluedroid Support & GATT support.

Contains:
- BLE Mesh and BLE coexistence support
- `CONFIG_BLE_MESH_GATT_PROXY_CLIENT`
- `CONFIG_BLE_MESH_GATT_PROXY_SERVER`
- BLE Mesh NET BUF DEBUG LOG LEVEL
- `CONFIG_BLE_MESH_PROV`
- `CONFIG_BLE_MESH_PROXY`
- BLE Mesh specific test option
- BLE Mesh STACK DEBUG LOG LEVEL
- `CONFIG_BLE_MESH_NO_LOG`
- `CONFIG_BLE_MESH_IVU_DIVIDER`
- `CONFIG_BLE_MESH_FAST_PROV`
- `CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC`
- `CONFIG_BLE_MESH_CRPL`
- `CONFIG_BLE_MESH_RX_SDU_MAX`
• CONFIG_BLE_MESH_MODEL_KEY_COUNT
• CONFIG_BLE_MESH_APP_KEY_COUNT
• CONFIG_BLE_MESH_MODEL_GROUP_COUNT
• CONFIG_BLE_MESH_LABEL_COUNT
• CONFIG_BLE_MESH_SUBNET_COUNT
• CONFIG_BLE_MESH_TX_SEG_MAX
• CONFIG_BLE_MESH_RX_SEG_MSG_COUNT
• CONFIG_BLE_MESH_TX_SEG_MSG_COUNT
• CONFIG_BLE_MESH_MEM_ALLOC_MODE
• CONFIG_BLE_MESH_MSG_CACHE_SIZE
• CONFIG_BLE_MESH_ADV_BUF_COUNT
• CONFIG_BLE_MESH_PB_GATT
• CONFIG_BLE_MESH_PB_ADV
• CONFIG_BLE_MESH_IVU_RECOVERY_IVI
• CONFIG_BLE_MESH_RELAY
• CONFIG_BLE_MESH_SETTINGS
• CONFIG_BLE_MESH_DEINIT
• CONFIG_BLE_MESH_USE_DUPLICATE_SCAN
• Support for BLE Mesh Client/Server models
• Support for BLE Mesh Foundation models
• CONFIG_BLE_MESH_NODE
• CONFIG_BLE_MESH_PROVISIONER
• CONFIG_BLE_MESH_FRIEND
• CONFIG_BLE_MESH_LOW_POWER
• CONFIG_BLE_MESH_HCI_5_0
• CONFIG_BLE_MESH_IV_UPDATE_TEST
• CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT

CONFIG_BLE_MESH_HCI_5_0

Support sending 20ms non-connectable adv packets

*Found in: Component config > CONFIG_BLE_MESH*

It is a temporary solution and needs further modifications.

*Default value:*

• Yes (enabled) if CONFIG_BLE_MESH

CONFIG_BLE_MESH_USE_DUPLICATE_SCAN

Support Duplicate Scan in BLE Mesh

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to allow using specific duplicate scan filter in BLE Mesh, and Scan Duplicate Type must be set by choosing the option in the Bluetooth Controller section in menuconfig, which is “Scan Duplicate By Device Address and Advertising Data”.

*Default value:*

• Yes (enabled) if BT_BLUEDROID_ENABLED && CONFIG_BLE_MESH

CONFIG_BLE_MESH_MEM_ALLOC_MODE

Memory allocation strategy

*Found in: Component config > CONFIG_BLE_MESH*

Allocation strategy for BLE Mesh stack, essentially provides ability to allocate all required dynamic allocations from,

• Internal DRAM memory only
Chapter 2. API

- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF
- Internal IRAM memory wherever applicable else internal DRAM

Recommended mode here is always internal (*), since that is most preferred from security perspective. But if application requirement does not allow sufficient free internal memory then alternate mode can be selected.

(*) In case of ESP32-S2/ESP32-S3, hardware allows encryption of external SPIRAM contents provided hardware flash encryption feature is enabled. In that case, using external SPIRAM allocation strategy is also safe choice from security perspective.

Available options:
- Internal DRAM (BLE_MESH_MEM_ALLOC_MODE_INTERNAL)
- External SPIRAM (BLE_MESH_MEM_ALLOC_MODE_EXTERNAL)
- Default alloc mode (BLE_MESH_MEM_ALLOC_MODE_DEFAULT)

Enable this option to use the default memory allocation strategy when external SPIRAM is enabled. See the SPIRAM options for more details.

- Internal IRAM (BLE_MESH_MEM_ALLOC_MODE_IRAM_8BIT)

Allows to use IRAM memory region as 8bit accessible region. Every unaligned (8bit or 16bit) access will result in an exception and incur penalty of certain clock cycles per unaligned read/write.

**CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC**

Enable FreeRTOS static allocation

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to use FreeRTOS static allocation APIs for BLE Mesh, which provides the ability to use different dynamic memory (i.e. SPIRAM or IRAM) for FreeRTOS objects. If this option is disabled, the FreeRTOS static allocation APIs will not be used, and internal DRAM will be allocated for FreeRTOS objects.

**Default value:**
- No (disabled) if (CONFIG_SPIRAM || CONFIG_ESP32_IRAM_AS_8BIT_ACCESSIBLE_MEMORY) && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC_MODE**

Memory allocation for FreeRTOS objects

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FREERTOS_STATIC_ALLOC*

Choose the memory to be used for FreeRTOS objects.

**Available options:**
- External SPIRAM (BLE_MESH_FREERTOS_STATIC_ALLOC_EXTERNAL)
  If enabled, BLE Mesh allocates dynamic memory from external SPIRAM for FreeRTOS objects, i.e. mutex, queue, and task stack. External SPIRAM can only be used for task stack when SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY is enabled. See the SPIRAM options for more details.
- Internal IRAM (BLE_MESH_FREERTOS_STATIC_ALLOC_IRAM_8BIT)
  If enabled, BLE Mesh allocates dynamic memory from internal IRAM for FreeRTOS objects, i.e. mutex, queue. Note: IRAM region cannot be used as task stack.

**CONFIG_BLE_MESH_DEINIT**

Support de-initialize BLE Mesh stack

*Found in: Component config > CONFIG_BLE_MESH*
If enabled, users can use the function esp_ble_mesh_deinit() to de-initialize the whole BLE Mesh stack.

**Default value:**
- Yes (enabled) if `CONFIG_BLE_MESH`

**BLE Mesh and BLE coexistence support**
Contains:
- `CONFIG_BLE_MESH_SUPPORT_BLE_SCAN`
- `CONFIG_BLE_MESH_SUPPORT_BLE_ADV`

**CONFIG_BLE_MESH_SUPPORT_BLE_ADV**
Support sending normal BLE advertising packets

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh and BLE coexistence support*

When selected, users can send normal BLE advertising packets with specific API.

**Default value:**
- No (disabled) if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_BLE_ADV_BUF_COUNT**
Number of advertising buffers for BLE advertising packets

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh and BLE coexistence support > CONFIG_BLE_MESH_SUPPORT_BLE_ADV*

Number of advertising buffers for BLE packets available.

**Range:**
- from 1 to 255 if `CONFIG_BLE_MESH_SUPPORT_BLE_ADV` && `CONFIG_BLE_MESH`

**Default value:**
- 3 if `CONFIG_BLE_MESH_SUPPORT_BLE_ADV` && `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_SUPPORT_BLE_SCAN**
Support scanning normal BLE advertising packets

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh and BLE coexistence support*

When selected, users can register a callback and receive normal BLE advertising packets in the application layer.

**Default value:**
- No (disabled) if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_FAST_PROV**
Enable BLE Mesh Fast Provisioning

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to allow BLE Mesh fast provisioning solution to be used. When there are multiple unprovisioned devices around, fast provisioning can greatly reduce the time consumption of the whole provisioning process. When this option is enabled, and after an unprovisioned device is provisioned into a node successfully, it can be changed to a temporary Provisioner.

**Default value:**
- No (disabled) if `CONFIG_BLE_MESH`
Chapter 2. API 参考

**CONFIG_BLE_MESH_NODE**

Support for BLE Mesh Node

*Found in: Component config > CONFIG_BLE_MESH*

Enable the device to be provisioned into a node. This option should be enabled when an unprovisioned device is going to be provisioned into a node and communicate with other nodes in the BLE Mesh network.

**CONFIG_BLE_MESH_PROVISIONER**

Support for BLE Mesh Provisioner

*Found in: Component config > CONFIG_BLE_MESH*

Enable the device to be a Provisioner. The option should be enabled when a device is going to act as a Provisioner and provision unprovisioned devices into the BLE Mesh network.

**CONFIG_BLE_MESH_WAIT_FOR_PROV_MAX_DEV_NUM**

Maximum number of unprovisioned devices that can be added to device queue

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many unprovisioned devices can be added to device queue for provisioning. Users can use this option to define the size of the queue in the bottom layer which is used to store unprovisioned device information (e.g. Device UUID, address).

*Range:*
- from 1 to 100 if `CONFIG_BLE_MESH_PROVISIONER && CONFIG_BLE_MESH`

*Default value:*
- 10 if `CONFIG_BLE_MESH_PROVISIONER && CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_MAX_PROV_NODES**

Maximum number of devices that can be provisioned by Provisioner

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many devices can be provisioned by a Provisioner. This value indicates the maximum number of unprovisioned devices which can be provisioned by a Provisioner. For instance, if the value is 6, it means the Provisioner can provision up to 6 unprovisioned devices. Theoretically a Provisioner without the limitation of its memory can provision up to 32766 unprovisioned devices, here we limit the maximum number to 100 just to limit the memory used by a Provisioner. The bigger the value is, the more memory it will cost by a Provisioner to store the information of nodes.

*Range:*
- from 1 to 1000 if `CONFIG_BLE_MESH_PROVISIONER && CONFIG_BLE_MESH`

*Default value:*
- 10 if `CONFIG_BLE_MESH_PROVISIONER && CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_PBASAME_TIME**

Maximum number of PB-ADV running at the same time by Provisioner

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many devices can be provisioned at the same time using PB-ADV. For examples, if the value is 2, it means a Provisioner can provision two unprovisioned devices with PB-ADV at the same time.

*Range:*

**Chapter 2. API**

- from 1 to 10 if `CONFIG_BLE_MESH_PB_ADV` & `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**Default value:**
- 2 if `CONFIG_BLE_MESH_PB_ADV` & `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_PBG_SAME_TIME**

Maximum number of PB-GATT running at the same time by Provisioner

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many devices can be provisioned at the same time using PB-GATT. For example, if the value is 2, it means a Provisioner can provision two unprovisioned devices with PB-GATT at the same time.

**Range:**
- from 1 to 5 if `CONFIG_BLE_MESH_PB_GATT` & `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**Default value:**
- 1 if `CONFIG_BLE_MESH_PB_GATT` & `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_PROVISIONER_SUBNET_COUNT**

Maximum number of mesh subnets that can be created by Provisioner

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many subnets per network a Provisioner can create. Indeed, this value decides the number of network keys which can be added by a Provisioner.

**Range:**
- from 1 to 4096 if `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**Default value:**
- 3 if `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_PROVISIONER_APP_KEY_COUNT**

Maximum number of application keys that can be owned by Provisioner

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER*

This option specifies how many application keys the Provisioner can have. Indeed, this value decides the number of the application keys which can be added by a Provisioner.

**Range:**
- from 1 to 4096 if `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**Default value:**
- 3 if `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_PROVISIONER_RECV_HB**

Support receiving Heartbeat messages

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER*

When this option is enabled, Provisioner can call specific functions to enable or disable receiving Heartbeat messages and notify them to the application layer.

**Default value:**
- No (disabled) if `CONFIG_BLE_MESH_PROVISIONER` & `CONFIG_BLE_MESH`
**CONFIG_BLE_MESH_PROVISIONER_RECV_HB_FILTER_SIZE**

Maximum number of filter entries for receiving Heartbeat messages

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER > CONFIG_BLE_MESH_PROVISIONER_RECV_HB*

This option specifies how many heartbeat filter entries Provisioner supports. The heartbeat filter (acceptlist or rejectlist) entries are used to store a list of SRC and DST which can be used to decide if a heartbeat message will be processed and notified to the application layer by Provisioner. Note: The filter is an empty rejectlist by default.

*Range:*
  - from 1 to 1000 if CONFIG_BLE_MESH_PROVISIONER_RECV_HB && CONFIG_BLE_MESH_PROVISIONER && CONFIG_BLE_MESH

*Default value:*
  - 3 if CONFIG_BLE_MESH_PROVISIONER_RECV_HB && CONFIG_BLE_MESH_PROVISIONER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_PROV**

BLE Mesh Provisioning support

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to support BLE Mesh Provisioning functionality. For BLE Mesh, this option should be always enabled.

*Default value:*
  - Yes (enabled) if CONFIG_BLE_MESH

**CONFIG_BLE_MESH_PB_ADV**

Provisioning support using the advertising bearer (PB-ADV)

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to allow the device to be provisioned over the advertising bearer. This option should be enabled if PB-ADV is going to be used during provisioning procedure.

*Default value:*
  - Yes (enabled) if CONFIG_BLE_MESH

**CONFIG_BLE_MESH_UNPROVISIONED_BEACON_INTERVAL**

Interval between two consecutive Unprovisioned Device Beacon

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PB_ADV*

This option specifies the interval of sending two consecutive unprovisioned device beacon, users can use this option to change the frequency of sending unprovisioned device beacon. For example, if the value is 5, it means the unprovisioned device beacon will send every 5 seconds. When the option of BLE_MESH_FAST_PROV is selected, the value is better to be 3 seconds, or less.

*Range:*
  - from 1 to 100 if CONFIG_BLE_MESH_NODE && CONFIG_BLE_MESH_PB_ADV && CONFIG_BLE_MESH

*Default value:*
  - 5 if CONFIG_BLE_MESH_NODE && CONFIG_BLE_MESH_PB_ADV && CONFIG_BLE_MESH
  - 3 if CONFIG_BLE_MESH_FAST_PROV && CONFIG_BLE_MESH_NODE && CONFIG_BLE_MESH_PB_ADV && CONFIG_BLE_MESH
CONFIG_BLE_MESH_PB_GATT
Provisioning support using GATT (PB-GATT)

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to allow the device to be provisioned over GATT. This option should be enabled if PB-GATT is going to be used during provisioning procedure.

# Virtual option enabled whenever any Proxy protocol is needed

CONFIG_BLE_MESH_PROXY
BLE Mesh Proxy protocol support

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to support BLE Mesh Proxy protocol used by PB-GATT and other proxy pdu transmission.

**Default value:**
- Yes (enabled) if CONFIG_BLE_MESH

CONFIG_BLE_MESH_GATT_PROXY_SERVER
BLE Mesh GATT Proxy Server

*Found in: Component config > CONFIG_BLE_MESH*

This option enables support for Mesh GATT Proxy Service, i.e. the ability to act as a proxy between a Mesh GATT Client and a Mesh network. This option should be enabled if a node is going to be a Proxy Server.

**Default value:**
- Yes (enabled) if CONFIG_BLE_MESH_NODE && CONFIG_BLE_MESH

CONFIG_BLE_MESH_NODE_ID_TIMEOUT
Node Identity advertising timeout

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_GATT_PROXY_SERVER*

This option determines for how long the local node advertises using Node Identity. The given value is in seconds. The specification limits this to 60 seconds and lists it as the recommended value as well. So leaving the default value is the safest option. When an unprovisioned device is provisioned successfully and becomes a node, it will start to advertise using Node Identity during the time set by this option. And after that, Network ID will be advertised.

**Range:**
- from 1 to 60 if CONFIG_BLE_MESH_GATT_PROXY_SERVER && CONFIG_BLE_MESH

**Default value:**
- 60 if CONFIG_BLE_MESH_GATT_PROXY_SERVER && CONFIG_BLE_MESH

CONFIG_BLE_MESH_PROXY_FILTER_SIZE
Maximum number of filter entries per Proxy Client

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_GATT_PROXY_SERVER*

This option specifies how many Proxy Filter entries the local node supports. The entries of Proxy filter (whitelist or blacklist) are used to store a list of addresses which can be used to decide which messages will be forwarded to the Proxy Client by the Proxy Server.

**Range:**
Chapter 2. API

- from 1 to 32767 if CONFIG_BLE_MESH_GATT_PROXY_SERVER && CONFIG_BLE_MESH

**Default value:**
- 4 if CONFIG_BLE_MESH_GATT_PROXY_SERVER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_GATT_PROXY_CLIENT**

BLE Mesh GATT Proxy Client

*Found in: Component config > CONFIG_BLE_MESH*

This option enables support for Mesh GATT Proxy Client. The Proxy Client can use the GATT bearer to send mesh messages to a node that supports the advertising bearer.

**Default value:**
- No (disabled) if CONFIG_BLE_MESH

**CONFIG_BLE_MESH_SETTINGS**

Store BLE Mesh configuration persistently

*Found in: Component config > CONFIG_BLE_MESH*

When selected, the BLE Mesh stack will take care of storing/restoring the BLE Mesh configuration persistently in flash. If the device is a BLE Mesh node, when this option is enabled, the configuration of the device will be stored persistently, including unicast address, NetKey, AppKey, etc. And if the device is a BLE Mesh Provisioner, the information of the device will be stored persistently, including the information of provisioned nodes, NetKey, AppKey, etc.

**Default value:**
- No (disabled) if CONFIG_BLE_MESH

**CONFIG_BLE_MESH_STORE_TIMEOUT**

Delay (in seconds) before storing anything persistently

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS*

This value defines in seconds how soon any pending changes are actually written into persistent storage (flash) after a change occurs. The option allows nodes to delay a certain period of time to save proper information to flash. The default value is 0, which means information will be stored immediately once there are updates.

**Range:**
- from 0 to 1000000 if CONFIG_BLE_MESH_SETTINGS && CONFIG_BLE_MESH

**Default value:**
- 0 if CONFIG_BLE_MESH_SETTINGS && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_SEQ_STORE_RATE**

How often the sequence number gets updated in storage

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS*

This value defines how often the local sequence number gets updated in persistent storage (i.e. flash), e.g. a value of 100 means that the sequence number will be stored to flash on every 100th increment. If the node sends messages very frequently a higher value makes more sense, whereas if the node sends infrequently a value as low as 0 (update storage for every increment) can make sense. When the stack gets initialized it will add sequence number to the last stored one, so that it starts off with a value that’s guaranteed to be larger than the last one used before power off.

**Range:**
- from 0 to 1000000 if CONFIG_BLE_MESH_SETTINGS && CONFIG_BLE_MESH
**Default value:**
- 0 if `CONFIG_BLE_MESH_SETTINGS` & `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_RPL_STORE_TIMEOUT**

Minimum frequency that the RPL gets updated in storage

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS*

This value defines in seconds how soon the RPL (Replay Protection List) gets written to persistent storage after a change occurs. If the node receives messages frequently, then a large value is recommended. If the node receives messages rarely, then the value can be as low as 0 (which means the RPL is written into the storage immediately). Note that if the node operates in a security-sensitive case, and there is a risk of sudden power-off, then a value of 0 is strongly recommended. Otherwise, a power loss before RPL being written into the storage may introduce message replay attacks and system security will be in a vulnerable state.

*Range:*
- from 0 to 1000000 if `CONFIG_BLE_MESH_SETTINGS` & `CONFIG_BLE_MESH`

*Default value:*
- 0 if `CONFIG_BLE_MESH_SETTINGS` & `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_SETTINGS_BACKWARD_COMPATIBILITY**

A specific option for settings backward compatibility

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS*

This option is created to solve the issue of failure in recovering node information after mesh stack updates. In the old version mesh stack, there is no key of “mesh/role” in nvs. In the new version mesh stack, key of “mesh/role” is added in nvs, recovering node information needs to check “mesh/role” key in nvs and implements selective recovery of mesh node information. Therefore, there may be failure in recovering node information during node restarting after OTA.

The new version mesh stack adds the option of “mesh/role” because we have added the support of storing Provisioner information, while the old version only supports storing node information.

If users are updating their nodes from old version to new version, we recommend enabling this option, so that system could set the flag in advance before recovering node information and make sure the node information recovering could work as expected.

*Default value:*
- No (disabled) if `CONFIG_BLE_MESH_NODE` & `CONFIG_BLE_MESH_SETTINGS` & `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_SPECIFIC_PARTITION**

Use a specific NVS partition for BLE Mesh

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS*

When selected, the mesh stack will use a specified NVS partition instead of default NVS partition. Note that the specified partition must be registered with NVS using `nvs_flash_init_partition()` API, and the partition must exists in the csv file. When Provisioner needs to store a large amount of nodes’ information in the flash (e.g. more than 20), this option is recommended to be enabled.

*Default value:*
- No (disabled) if `CONFIG_BLE_MESH_SETTINGS` & `CONFIG_BLE_MESH`
**CONFIG_BLE_MESH_PARTITION_NAME**

Name of the NVS partition for BLE Mesh

*Found in:* Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS > CONFIG_BLE_MESH_SPECIFIC_PARTITION

This value defines the name of the specified NVS partition used by the mesh stack.

*Default value:*
- “ble_mesh” if CONFIG_BLE_MESH_SPECIFIC_PARTITION && CONFIG_BLE_MESH_SETTINGS && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE**

Support using multiple NVS namespaces by Provisioner

*Found in:* Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

When selected, Provisioner can use different NVS namespaces to store different instances of mesh information. For example, if in the first room, Provisioner uses NetKey A, AppKey A and provisions three devices, these information will be treated as mesh information instance A. When the Provisioner moves to the second room, it uses NetKey B, AppKey B and provisions two devices, then the information will be treated as mesh information instance B. Here instance A and instance B will be stored in different namespaces. With this option enabled, Provisioner needs to use specific functions to open the corresponding NVS namespace, restore the mesh information, release the mesh information or erase the mesh information.

*Default value:*
- No (disabled) if CONFIG_BLE_MESH_PROVISIONER && CONFIG_BLE_MESH_SETTINGS && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_MAX_NVS_NAMESPACE**

Maximum number of NVS namespaces

*Found in:* Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS > CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE

This option specifies the maximum NVS namespaces supported by Provisioner.

*Range:*
- from 1 to 255 if CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE && CONFIG_BLE_MESH_SETTINGS && CONFIG_BLE_MESH

*Default value:*
- 2 if CONFIG_BLE_MESH_USE_MULTIPLE_NAMESPACE && CONFIG_BLE_MESH_SETTINGS && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_SUBNET_COUNT**

Maximum number of mesh subnets per network

*Found in:* Component config > CONFIG_BLE_MESH

This option specifies how many subnets a Mesh network can have at the same time. Indeed, this value decides the number of the network keys which can be owned by a node.

*Range:*
- from 1 to 4096 if CONFIG_BLE_MESH

*Default value:*
- 3 if CONFIG_BLE_MESH
**CONFIG_BLE_MESH_APP_KEY_COUNT**

Maximum number of application keys per network

*Found in: Component config > CONFIG_BLE_MESH*

This option specifies how many application keys the device can store per network. Indeed, this value decides the number of the application keys which can be owned by a node.

*Range:*
- from 1 to 4096 if `CONFIG_BLE_MESH`

*Default value:*
- 3 if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_MODEL_KEY_COUNT**

Maximum number of application keys per model

*Found in: Component config > CONFIG_BLE_MESH*

This option specifies the maximum number of application keys to which each model can be bound.

*Range:*
- from 1 to 4096 if `CONFIG_BLE_MESH`

*Default value:*
- 3 if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_MODEL_GROUP_COUNT**

Maximum number of group address subscriptions per model

*Found in: Component config > CONFIG_BLE_MESH*

This option specifies the maximum number of addresses to which each model can be subscribed.

*Range:*
- from 1 to 4096 if `CONFIG_BLE_MESH`

*Default value:*
- 3 if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_LABEL_COUNT**

Maximum number of Label UUIDs used for Virtual Addresses

*Found in: Component config > CONFIG_BLE_MESH*

This option specifies how many Label UUIDs can be stored. Indeed, this value decides the number of the Virtual Addresses that can be supported by a node.

*Range:*
- from 0 to 4096 if `CONFIG_BLE_MESH`

*Default value:*
- 3 if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_CRPL**

Maximum capacity of the replay protection list

*Found in: Component config > CONFIG_BLE_MESH*

This option specifies the maximum capacity of the replay protection list. It is similar to Network message cache size, but has a different purpose. The replay protection list is used to prevent a node from replay attack, which will store the source address and sequence number of the received mesh messages. For Provisioner, the replay protection list size should not be smaller than the maximum number of nodes whose information can be stored. And the element number of each node should also be taken into account.
consideration. For example, if Provisioner can provision up to 20 nodes and each node contains two elements, then the replay protection list size of Provisioner should be at least 40.

Range:
- from 2 to 65535 if `CONFIG_BLE_MESH`

Default value:
- 10 if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_MSG_CACHE_SIZE**

Network message cache size

*Found in: Component config > CONFIG_BLE_MESH*

Number of messages that are cached for the network. This helps prevent unnecessary decryption operations and unnecessary relays. This option is similar to Replay protection list, but has a different purpose. A node is not required to cache the entire Network PDU and may cache only part of it for tracking, such as values for SRC/SEQ or others.

Range:
- from 2 to 65535 if `CONFIG_BLE_MESH`

Default value:
- 10 if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_ADV_BUF_COUNT**

Number of advertising buffers

*Found in: Component config > CONFIG_BLE_MESH*

Number of advertising buffers available. The transport layer reserves ADV_BUF_COUNT - 3 buffers for outgoing segments. The maximum outgoing SDU size is 12 times this value (out of which 4 or 8 bytes are used for the Transport Layer MIC). For example, 5 segments means the maximum SDU size is 60 bytes, which leaves 56 bytes for application layer data using a 4-byte MIC, or 52 bytes using an 8-byte MIC.

Range:
- from 6 to 256 if `CONFIG_BLE_MESH`

Default value:
- 60 if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_IVU_DIVIDER**

Divider for IV Update state refresh timer

*Found in: Component config > CONFIG_BLE_MESH*

When the IV Update state enters Normal operation or IV Update in Progress, we need to keep track of how many hours has passed in the state, since the specification requires us to remain in the state at least for 96 hours (Update in Progress has an additional upper limit of 144 hours).

In order to fulfill the above requirement, even if the node might be powered off once in a while, we need to store persistently how many hours the node has been in the state. This doesn’t necessarily need to happen every hour (thanks to the flexible duration range). The exact cadence will depend a lot on the ways that the node will be used and what kind of power source it has.

Since there is no single optimal answer, this configuration option allows specifying a divider, i.e. how many intervals the 96 hour minimum gets split into. After each interval the duration that the node has been in the current state gets stored to flash. E.g. the default value of 4 means that the state is saved every 24 hours (96 / 4).

Range:
- from 2 to 96 if `CONFIG_BLE_MESH`
**CONFIG_BLE_MESH_IVU_RECOVERY_IVI**

Recovery the IV index when the latest whole IV update procedure is missed

*Found in: Component config > CONFIG_BLE_MESH*

According to Section 3.10.5 of Mesh Specification v1.0.1. If a node in Normal Operation receives a Secure Network beacon with an IV index equal to the last known IV index+1 and the IV Update Flag set to 0, the node may update its IV without going to the IV Update in Progress state, or it may initiate an IV Index Recovery procedure (Section 3.10.6), or it may ignore the Secure Network beacon. The node makes the choice depending on the time since last IV update and the likelihood that the node has missed the Secure Network beacons with the IV update Flag. When the above situation is encountered, this option can be used to decide whether to perform the IV index recovery procedure.

*Default value:*

- No (disabled) if `CONFIG_BLE_MESH`

---

**CONFIG_BLE_MESH_TX_SEG_MSG_COUNT**

Maximum number of simultaneous outgoing segmented messages

*Found in: Component config > CONFIG_BLE_MESH*

Maximum number of simultaneous outgoing multi-segment and/or reliable messages. The default value is 1, which means the device can only send one segmented message at a time. And if another segmented message is going to be sent, it should wait for the completion of the previous one. If users are going to send multiple segmented messages at the same time, this value should be configured properly.

*Range:*

- from 1 to 255 if `CONFIG_BLE_MESH`

*Default value:*

- 1 if `CONFIG_BLE_MESH`

---

**CONFIG_BLE_MESH_RX_SEG_MSG_COUNT**

Maximum number of simultaneous incoming segmented messages

*Found in: Component config > CONFIG_BLE_MESH*

Maximum number of simultaneous incoming multi-segment and/or reliable messages. The default value is 1, which means the device can only receive one segmented message at a time. And if another segmented message is going to be received, it should wait for the completion of the previous one. If users are going to receive multiple segmented messages at the same time, this value should be configured properly.

*Range:*

- from 1 to 255 if `CONFIG_BLE_MESH`

*Default value:*

- 1 if `CONFIG_BLE_MESH`

---

**CONFIG_BLE_MESH_RX_SDU_MAX**

Maximum incoming Upper Transport Access PDU length

*Found in: Component config > CONFIG_BLE_MESH*

Maximum incoming Upper Transport Access PDU length. Leave this to the default value, unless you really need to optimize memory usage.

*Range:*

- from CONFIG_BLE_MESH to CONFIG_BLE_MESH
Chapter 2. API

• from 36 to 384 if CONFIG_BLE_MESH

Default value:
• 384 if CONFIG_BLE_MESH

CONFIG_BLE_MESH_TX_SEG_MAX

Maximum number of segments in outgoing messages

Found in: Component config > CONFIG_BLE_MESH

Maximum number of segments supported for outgoing messages. This value should typically be fine-tuned based on what models the local node supports, i.e. what’s the largest message payload that the node needs to be able to send. This value affects memory and call stack consumption, which is why the default is lower than the maximum that the specification would allow (32 segments).

The maximum outgoing SDU size is 12 times this number (out of which 4 or 8 bytes is used for the Transport Layer MIC). For example, 5 segments means the maximum SDU size is 60 bytes, which leaves 56 bytes for application layer data using a 4-byte MIC and 52 bytes using an 8-byte MIC.

Be sure to specify a sufficient number of advertising buffers when setting this option to a higher value. There must be at least three more advertising buffers (BLE_MESH_ADV_BUF_COUNT) as there are outgoing segments.

Range:
• from 2 to 32 if CONFIG_BLE_MESH

Default value:
• 32 if CONFIG_BLE_MESH

CONFIG_BLE_MESH_RELAY

Relay support

Found in: Component config > CONFIG_BLE_MESH

Support for acting as a Mesh Relay Node. Enabling this option will allow a node to support the Relay feature, and the Relay feature can still be enabled or disabled by proper configuration messages. Disabling this option will let a node not support the Relay feature.

Default value:
• Yes (enabled) if CONFIG_BLE_MESH_NODE && CONFIG_BLE_MESH

CONFIG_BLE_MESH_RELAY_ADV_BUF

Use separate advertising buffers for relay packets

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_RELAY

When selected, self-send packets will be put in a high-priority queue and relay packets will be put in a low-priority queue.

Default value:
• No (disabled) if CONFIG_BLE_MESH_RELAY && CONFIG_BLE_MESH

CONFIG_BLE_MESH_RELAY_ADV_BUF_COUNT

Number of advertising buffers for relay packets

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_RELAY > CONFIG_BLE_MESH_RELAY_ADV_BUF

Number of advertising buffers for relay packets available.

Range:
Chapter 2. API

• from 6 to 256 if CONFIG_BLE_MESH_RELAY_ADV_BUF && CONFIG_BLE_MESH_RELAY && CONFIG_BLE_MESH

**Default value:**
• 60 if CONFIG_BLE_MESH_RELAY_ADV_BUF && CONFIG_BLE_MESH_RELAY && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_LOW_POWER**

Support for Low Power features

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to operate as a Low Power Node. If low power consumption is required by a node, this option should be enabled. And once the node enters the mesh network, it will try to find a Friend node and establish a friendship.

**CONFIG_BLE_MESH_LPN_ESTABLISHMENT**

Perform Friendship establishment using low power

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*

Perform the Friendship establishment using low power with the help of a reduced scan duty cycle. The downside of this is that the node may miss out on messages intended for it until it has successfully set up Friendship with a Friend node. When this option is enabled, the node will stop scanning for a period of time after a Friend Request or Friend Poll is sent, so as to reduce more power consumption.

**Default value:**
• No (disabled) if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_LPN_AUTO**

Automatically start looking for Friend nodes once provisioned

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*

Once provisioned, automatically enable LPN functionality and start looking for Friend nodes. If this option is disabled LPN mode needs to be manually enabled by calling bt_mesh_lpn_set(true). When an un provisioned device is provisioned successfully and becomes a node, enabling this option will trigger the node starts to send Friend Request at a certain period until it finds a proper Friend node.

**Default value:**
• No (disabled) if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_LPN_AUTO_TIMEOUT**

Time from last received message before going to LPN mode

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER > CONFIG_BLE_MESH_LPN_AUTO*

Time in seconds from the last received message, that the node waits out before starting to look for Friend nodes.

**Range:**
• from 0 to 3600 if CONFIG_BLE_MESH_LPN_AUTO && CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**Default value:**
• 15 if CONFIG_BLE_MESH_LPN_AUTO && CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH
CONFIG_BLE_MESH_LPN_RETRY_TIMEOUT

Retry timeout for Friend requests

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Time in seconds between Friend Requests, if a previous Friend Request did not yield any acceptable Friend Offers.

Range:
- from 1 to 3600 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

Default value:
- 6 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

CONFIG_BLE_MESH_LPN_RSSI_FACTOR

RSSIFactor, used in Friend Offer Delay calculation

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The contribution of the RSSI, measured by the Friend node, used in Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5. RSSIFactor, one of the parameters carried by Friend Request sent by Low Power node, which is used to calculate the Friend Offer Delay.

Range:
- from 0 to 3 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

Default value:
- 0 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

CONFIG_BLE_MESH_LPN_RECV_WIN_FACTOR

ReceiveWindowFactor, used in Friend Offer Delay calculation

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The contribution of the supported Receive Window used in Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5. ReceiveWindowFactor, one of the parameters carried by Friend Request sent by Low Power node, which is used to calculate the Friend Offer Delay.

Range:
- from 0 to 3 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

Default value:
- 0 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

CONFIG_BLE_MESH_LPN_MIN_QUEUE_SIZE

Minimum size of the acceptable friend queue (MinQueueSizeLog)

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The MinQueueSizeLog field is defined as log_2(N), where N is the minimum number of maximum size Lower Transport PDUs that the Friend node can store in its Friend Queue. As an example, MinQueueSizeLog value 1 gives N = 2, and value 7 gives N = 128.

Range:
- from 1 to 7 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

Default value:
- 1 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

CONFIG_BLE_MESH_LPN_RECV_DELAY

Espressif Systems 1414 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
Receive delay requested by the local node

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*

The ReceiveDelay is the time between the Low Power node sending a request and listening for a response. This delay allows the Friend node time to prepare the response. The value is in units of milliseconds.

**Range:**
- from 10 to 255 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**Default value:**
- 100 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_LPN_POLL_TIMEOUT**

The value of the PollTimeout timer

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*

PollTimeout timer is used to measure time between two consecutive requests sent by a Low Power node. If no requests are received the Friend node before the PollTimeout timer expires, then the friendship is considered terminated. The value is in units of 100 milliseconds, so e.g. a value of 300 means 30 seconds. The smaller the value, the faster the Low Power node tries to get messages from corresponding Friend node and vice versa.

**Range:**
- from 10 to 244735 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**Default value:**
- 300 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_LPN_INIT_POLL_TIMEOUT**

The starting value of the PollTimeout timer

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*

The initial value of the PollTimeout timer when Friendship is to be established for the first time. After this, the timeout gradually grows toward the actual PollTimeout, doubling in value for each iteration. The value is in units of 100 milliseconds, so e.g. a value of 300 means 30 seconds.

**Range:**
- from 10 to if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**Default value:**
- if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_LPN_SCAN_LATENCY**

Latency for enabling scanning

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*

Latency (in milliseconds) is the time it takes to enable scanning. In practice, it means how much time in advance of the Receive Window, the request to enable scanning is made.

**Range:**
- from 0 to 50 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**Default value:**
- 10 if CONFIG_BLE_MESH_LOW_POWER && CONFIG_BLE_MESH

**CONFIG_BLE_MESH_LPN_GROUPS**

Number of groups the LPN can subscribe to

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*
Maximum number of groups to which the LPN can subscribe.

**Range:**
- from 0 to 16384 if `CONFIG_BLE_MESH_LOW_POWER` && `CONFIG_BLE_MESH`  
**Default value:**
- 8 if `CONFIG_BLE_MESH_LOW_POWER` && `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_LPN_SUB_ALL_NODES_ADDR**
Automatically subscribe all nodes address

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER*

Automatically subscribe all nodes address when friendship established.

**Default value:**
- No (disabled) if `CONFIG_BLE_MESH_LOW_POWER` && `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_FRIEND**
Support for Friend feature

*Found in: Component config > CONFIG_BLE_MESH*

Enable this option to be able to act as a Friend Node.

**CONFIG_BLE_MESH_FRIEND_RECV_WIN**
Friend Receive Window

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND*

Receive Window in milliseconds supported by the Friend node.

**Range:**
- from 1 to 255 if `CONFIG_BLE_MESH_FRIEND` && `CONFIG_BLE_MESH`  
**Default value:**
- 255 if `CONFIG_BLE_MESH_FRIEND` && `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_FRIEND_QUEUE_SIZE**
Minimum number of buffers supported per Friend Queue

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND*

Minimum number of buffers available to be stored for each local Friend Queue. This option decides the size of each buffer which can be used by a Friend node to store messages for each Low Power node.

**Range:**
- from 2 to 65536 if `CONFIG_BLE_MESH_FRIEND` && `CONFIG_BLE_MESH`  
**Default value:**
- 16 if `CONFIG_BLE_MESH_FRIEND` && `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_FRIEND_SUB_LIST_SIZE**
Friend Subscription List Size

*Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND*

Size of the Subscription List that can be supported by a Friend node for a Low Power node. And Low Power node can send Friend Subscription List Add or Friend Subscription List Remove messages to the Friend node to add or remove subscription addresses.

**Range:**
- from 0 to 1023 if `CONFIG_BLE_MESH_FRIEND` && `CONFIG_BLE_MESH`
Default value:
  • 3 if CONFIG_BLE_MESH_FRIEND && CONFIG_BLE_MESH

CONFIG_BLE_MESH_FRIEND_LPN_COUNT

Number of supported LPN nodes

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Number of Low Power Nodes with which a Friend can have Friendship simultaneously. A Friend node can have friendship with multiple Low Power nodes at the same time, while a Low Power node can only establish friendship with only one Friend node at the same time.

Range:
  • from 1 to 1000 if CONFIG_BLE_MESH_FRIEND && CONFIG_BLE_MESH

Default value:
  • 2 if CONFIG_BLE_MESH_FRIEND && CONFIG_BLE_MESH

CONFIG_BLE_MESH_FRIEND_SEG_RX

Number of incomplete segment lists per LPN

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Number of incomplete segment lists tracked for each Friends’ LPN. In other words, this determines from how many elements can segmented messages destined for the Friend queue be received simultaneously.

Range:
  • from 1 to 1000 if CONFIG_BLE_MESH_FRIEND && CONFIG_BLE_MESH

Default value:
  • 1 if CONFIG_BLE_MESH_FRIEND && CONFIG_BLE_MESH

CONFIG_BLE_MESH_NO_LOG

Disable BLE Mesh debug logs (minimize bin size)

Found in: Component config > CONFIG_BLE_MESH

Select this to save the BLE Mesh related rodata code size. Enabling this option will disable the output of BLE Mesh debug log.

Default value:
  • No (disabled) if CONFIG_BLE_MESH && CONFIG_BLE_MESH

BLE Mesh STACK DEBUG LOG LEVEL  Contains:
  • CONFIG_BLE_MESH_STACK_TRACE_LEVEL

CONFIG_BLE_MESH_STACK_TRACE_LEVEL

BLE_MESH_STACK

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh STACK DEBUG LOG LEVEL

Define BLE Mesh trace level for BLE Mesh stack.

Available options:
  • NONE (BLE_MESH_TRACE_LEVEL_NONE)
  • ERROR (BLE_MESH_TRACE_LEVEL_ERROR)
  • WARNING (BLE_MESH_TRACE_LEVEL_WARNING)
  • INFO (BLE_MESH_TRACE_LEVEL_INFO)
  • DEBUG (BLE_MESH_TRACE_LEVEL_DEBUG)
  • VERBOSE (BLE_MESH_TRACE_LEVEL_VERBOSE)
Chapter 2. API

**BLE Mesh NET BUF DEBUG LOG LEVEL** Contains:

- `CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL`

**CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL**

`BLE_MESH_NET_BUF`

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh NET BUF DEBUG LOG LEVEL*

Define BLE Mesh trace level for BLE Mesh net buffer.

*Available options:*

- **NONE (BLE_MESH_NET_BUF_TRACE_LEVEL_NONE)**
- **ERROR (BLE_MESH_NET_BUF_TRACE_LEVEL_ERROR)**
- **WARNING (BLE_MESH_NET_BUF_TRACE_LEVEL_WARNING)**
- **INFO (BLE_MESH_NET_BUF_TRACE_LEVEL_INFO)**
- **DEBUG (BLE_MESH_NET_BUF_TRACE_LEVEL_DEBUG)**
- **VERBOSE (BLE_MESH_NET_BUF_TRACE_LEVEL_VERBOSE)**

**CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT**

Timeout(ms) for client message response

*Found in: Component config > CONFIG_BLE_MESH*

Timeout value used by the node to get response of the acknowledged message which is sent by the client model. This value indicates the maximum time that a client model waits for the response of the sent acknowledged messages. If a client model uses 0 as the timeout value when sending acknowledged messages, then the default value will be used which is four seconds.

*Range:*

- **from 100 to 1200000 if CONFIG_BLE_MESH**

*Default value:*

- **4000 if CONFIG_BLE_MESH**

**Support for BLE Mesh Foundation models** Contains:

- `CONFIG_BLE_MESH_CFG_CLI`
- `CONFIG_BLE_MESH_HEALTH_CLI`
- `CONFIG_BLE_MESH_HEALTH_SRV`

**CONFIG_BLE_MESH_CFG_CLI**

Configuration Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models*

Enable support for Configuration Client model.

**CONFIG_BLE_MESH_HEALTH_CLI**

Health Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models*

Enable support for Health Client model.
CONFIG_BLE_MESH_HEALTH_SRV

Health Server model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Foundation models

Enable support for Health Server model.

Default value:
- Yes (enabled) if CONFIG_BLE_MESH

Support for BLE Mesh Client/Server models Contains:
- CONFIG_BLE_MESH_GENERIC_BATTERY_CLI
- CONFIG_BLE_MESH_GENERIC_DEF_TRANS_TIME_CLI
- CONFIG_BLE_MESH_GENERIC_LEVEL_CLI
- CONFIG_BLE_MESH_GENERIC_LOCATION_CLI
- CONFIG_BLE_MESH_GENERIC_ONOFF_CLI
- CONFIG_BLE_MESH_GENERIC_POWER_LEVEL_CLI
- CONFIG_BLE_MESH_GENERIC_POWER_ONOFF_CLI
- CONFIG_BLE_MESH_GENERIC_PROPERTY_CLI
- CONFIG_BLE_MESH_GENERIC_SERVER
- CONFIG_BLE_MESH_LIGHT_CTL_CLI
- CONFIG_BLE_MESH_LIGHT_HSL_CLI
- CONFIG_BLE_MESH_LIGHT_LC_CLI
- CONFIG_BLE_MESH_LIGHT_LIGHTNESS_CLI
- CONFIG_BLE_MESH_LIGHT_XYL_CLI
- CONFIG_BLE_MESH_LIGHTING_SERVER
- CONFIG_BLE_MESH_SCENE_CLI
- CONFIG_BLE_MESH_SCHEDULER_CLI
- CONFIG_BLE_MESH_SENSOR_CLI
- CONFIG_BLE_MESH_SENSOR_SERVER
- CONFIG_BLE_MESH_TIME_SCENE_SERVER
- CONFIG_BLE_MESH_TIME_CLI

CONFIG_BLE_MESH_GENERIC_ONOFF_CLI

Generic OnOff Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic OnOff Client model.

CONFIG_BLE_MESH GENERIC_LEVEL_CLI

Generic Level Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Level Client model.

CONFIG_BLE_MESH GENERIC DEF_TRANS_TIME_CLI

Generic Default Transition Time Client model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models

Enable support for Generic Default Transition Time Client model.
CONFIG_BLE_MESH_GENERIC_POWER_ONOFF_CLI
Generic Power OnOff Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Generic Power OnOff Client model.

CONFIG_BLE_MESH_GENERIC_POWER_LEVEL_CLI
Generic Power Level Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Generic Power Level Client model.

CONFIG_BLE_MESH_GENERIC_BATTERY_CLI
Generic Battery Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Generic Battery Client model.

CONFIG_BLE_MESH_GENERIC_LOCATION_CLI
Generic Location Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Generic Location Client model.

CONFIG_BLE_MESH_GENERIC_PROPERTY_CLI
Generic Property Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Generic Property Client model.

CONFIG_BLE_MESHSENSOR_CLI
Sensor Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Sensor Client model.

CONFIG_BLE_MESH_TIME_CLI
Time Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Time Client model.

CONFIG_BLE_MESH_SCENE_CLI
Scene Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*
Enable support for Scene Client model.
**CONFIG_BLE_MESH_SCHEDULER_CLI**

Scheduler Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Scheduler Client model.

**CONFIG_BLE_MESH_LIGHT_LIGHTNESS_CLI**

Light Lightness Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Light Lightness Client model.

**CONFIG_BLE_MESH_LIGHT_CTL_CLI**

Light CTL Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Light CTL Client model.

**CONFIG_BLE_MESH_LIGHT_HSL_CLI**

Light HSL Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Light HSL Client model.

**CONFIG_BLE_MESH_LIGHT_XYL_CLI**

Light XYL Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Light XYL Client model.

**CONFIG_BLE_MESH_LIGHT_LC_CLI**

Light LC Client model

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Light LC Client model.

**CONFIG_BLE_MESH GENERIC_SERVER**

Generic server models

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Generic server models.

**Default value:**

- Yes (enabled) if `CONFIG_BLE_MESH`
**CONFIG_BLE_MESH_SENSOR_SERVER**

Sensor server models

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Sensor server models.

**Default value:**
- Yes (enabled) if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_TIME_SCENE_SERVER**

Time and Scenes server models

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Time and Scenes server models.

**Default value:**
- Yes (enabled) if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_LIGHTING_SERVER**

Lighting server models

*Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client/Server models*

Enable support for Lighting server models.

**Default value:**
- Yes (enabled) if `CONFIG_BLE_MESH`

**CONFIG_BLE_MESH_IV_UPDATE_TEST**

Test the IV Update Procedure

*Found in: Component config > CONFIG_BLE_MESH*

This option removes the 96 hour limit of the IV Update Procedure and lets the state to be changed at any time. If IV Update test mode is going to be used, this option should be enabled.

**Default value:**
- No (disabled) if `CONFIG_BLE_MESH`

**BLE Mesh specific test option**  Contains:

- `CONFIG_BLE_MESH_DEBUG`
- `CONFIG_BLE_MESH_SHELL`
- `CONFIG_BLE_MESH_SELF_TEST`

**CONFIG_BLE_MESH_SELF_TEST**

Perform BLE Mesh self-tests

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option*

This option adds extra self-tests which are run every time BLE Mesh networking is initialized.

**Default value:**
- No (disabled) if `CONFIG_BLE_MESH`
CONFIG_BLE_MESH_TEST_AUTO_ENTER_NETWORK

Unprovisioned device enters mesh network automatically

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_SELF_TEST*

With this option enabled, an unprovisioned device can automatically enter the mesh network using a specific test function without the provisioning procedure. And on the Provisioner side, a test function needs to be invoked to add the node information into the mesh stack.

**Default value:**
- Yes (enabled) if CONFIG_BLE_MESH_SELF_TEST && CONFIG_BLE_MESH

CONFIG_BLE_MESH_TEST_USE_WHITE_LIST

Use white list to filter mesh advertising packets

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_SELF_TEST*

With this option enabled, users can use white list to filter mesh advertising packets while scanning.

**Default value:**
- No (disabled) if CONFIG_BLE_MESH_SELF_TEST && CONFIG_BLE_MESH

CONFIG_BLE_MESH_SHELL

Enable BLE Mesh shell

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option*

Activate shell module that provides BLE Mesh commands to the console.

**Default value:**
- No (disabled) if CONFIG_BLE_MESH

CONFIG_BLE_MESH_DEBUG

Enable BLE Mesh debug logs

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option*

Enable debug logs for the BLE Mesh functionality.

**Default value:**
- No (disabled) if CONFIG_BLE_MESH

CONFIG_BLE_MESH_DEBUG_NET

Network layer debug

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG*

Enable Network layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_TRANS

Transport layer debug

*Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG*

Enable Transport layer debug logs for the BLE Mesh functionality.
CONFIG_BLE_MESH_DEBUG_BEACON

Beacon debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Beacon-related debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_CRYPTO

Crypto debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable cryptographic debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_PROV

Provisioning debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Provisioning debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_ACCESS

Access layer debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Access layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_MODEL

Foundation model debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Foundation Models debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_ADV

Advertising debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable advertising debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_LOW_POWER

Low Power debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Low Power debug logs for the BLE Mesh functionality.
CONFIG_BLE_MESH_DEBUG_FRIEND

Friend debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Friend debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_PROXY

Proxy debug

*Found in:* Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Proxy protocol debug logs for the BLE Mesh functionality.

Driver Configurations  Contains:

- DAC Configuration
- GPIO Configuration
- GPTimer Configuration
- I2S Configuration
- Legacy ADC Configuration
- MCPWM Configuration
- PCNT Configuration
- RMT Configuration
- Sigma Delta Modulator Configuration
- SPI Configuration
- Temperature sensor Configuration
- TWAI Configuration
- UART Configuration

Legacy ADC Configuration  Contains:

- CONFIG_ADC_DISABLE_DAC
- Legacy ADC Calibration Configuration
- CONFIG_ADC_SUPPRESS_DEPRECATE_WARN

CONFIG_ADC_DISABLE_DAC

Disable DAC when ADC2 is used on GPIO 25 and 26

*Found in:* Component config > Driver Configurations > Legacy ADC Configuration

If this is set, the ADC2 driver will disable the output of the DAC corresponding to the specified channel. This is the default value.

For testing, disable this option so that we can measure the output of DAC by internal ADC.

**Default value:**

- Yes (enabled)

CONFIG_ADC_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

*Found in:* Component config > Driver Configurations > Legacy ADC Configuration

Weather to suppress the deprecation warnings when using legacy adc driver (driver/adc.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.
**Legacy ADC Calibration Configuration**  
Contains:

- `CONFIG_ADC_CALI_SUPPRESS_DEPRECATE_WARN`
- `CONFIG_ADC_CAL_EFUSE_VREF_ENABLE`
- `CONFIG_ADC_CAL_LUT_ENABLE`
- `CONFIG_ADC_CAL_EFUSE_TP_ENABLE`

**CONFIG_ADC_CAL_EFUSE_TP_ENABLE**

Use Two Point Values

*Found in: Component config > Driver Configurations > Legacy ADC Configuration > Legacy ADC Calibration Configuration*

Some ESP32s have Two Point calibration values burned into eFuse BLOCK3. This option will allow the ADC calibration component to characterize the ADC-Voltage curve using Two Point values if they are available.

**Default value:**

- Yes (enabled)

**CONFIG_ADC_CAL_EFUSE_VREF_ENABLE**

Use eFuse Vref

*Found in: Component config > Driver Configurations > Legacy ADC Configuration > Legacy ADC Calibration Configuration*

Some ESP32s have Vref burned into eFuse BLOCK0. This option will allow the ADC calibration component to characterize the ADC-Voltage curve using eFuse Vref if it is available.

**Default value:**

- Yes (enabled)

**CONFIG_ADC_CAL_LUT_ENABLE**

Use Lookup Tables

*Found in: Component config > Driver Configurations > Legacy ADC Configuration > Legacy ADC Calibration Configuration*

This option will allow the ADC calibration component to use Lookup Tables to correct for non-linear behavior in 11db attenuation. Other attenuations do not exhibit non-linear behavior hence will not be affected by this option.

**Default value:**

- Yes (enabled)

**CONFIG_ADC_CALI_SUPPRESS_DEPRECATE_WARN**

Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > Legacy ADC Configuration > Legacy ADC Calibration Configuration*

Whether to suppress the deprecation warnings when using legacy adc calibration driver (esp_adc_cal.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
**SPI Configuration**

Contains:

- `CONFIG_SPI_MASTER_ISR_IN_IRAM`
- `CONFIG_SPI_SLAVE_ISR_IN_IRAM`
- `CONFIG_SPI_MASTER_IN_IRAM`
- `CONFIG_SPI_SLAVE_IN_IRAM`

**CONFIG_SPI_MASTER_IN_IRAM**

Place transmitting functions of SPI master into IRAM

*Found in: Component config > Driver Configurations > SPI Configuration*

Normally only the ISR of SPI master is placed in the IRAM, so that it can work without the flash when interrupt is triggered. For other functions, there’s some possibility that the flash cache miss when running inside and out of SPI functions, which may increase the interval of SPI transactions. Enable this to put `queue_trans`, `get_trans_result` and `transmit` functions into the IRAM to avoid possible cache miss.

During unit test, this is enabled to measure the ideal case of api.

**Default value:**

- No (disabled)

**CONFIG_SPI_MASTER_ISR_IN_IRAM**

Place SPI master ISR function into IRAM

*Found in: Component config > Driver Configurations > SPI Configuration*

Place the SPI master ISR in to IRAM to avoid possible cache miss.

Also you can forbid the ISR being disabled during flash writing access, by add `ESP_INTR_FLAG_IRAM` when initializing the driver.

**Default value:**

- Yes (enabled)

**CONFIG_SPI_SLAVE_IN_IRAM**

Place transmitting functions of SPI slave into IRAM

*Found in: Component config > Driver Configurations > SPI Configuration*

Normally only the ISR of SPI slave is placed in the IRAM, so that it can work without the flash when interrupt is triggered. For other functions, there’s some possibility that the flash cache miss when running inside and out of SPI functions, which may increase the interval of SPI transactions. Enable this to put `queue_trans`, `get_trans_result` and `transmit` functions into the IRAM to avoid possible cache miss.

**Default value:**

- No (disabled)

**CONFIG_SPI_SLAVE_ISR_IN_IRAM**

Place SPI slave ISR function into IRAM

*Found in: Component config > Driver Configurations > SPI Configuration*

Place the SPI slave ISR in to IRAM to avoid possible cache miss.
Also you can forbid the ISR being disabled during flash writing access, by add ESP_INTR_FLAG_IRAM when initializing the driver.

Default value:
- Yes (enabled)

**TWAI Configuration**  Contains:
- `CONFIG_TWAI_ERRATA_FIX_RX_FRAME_INVALID`
- `CONFIG_TWAI_ERRATA_FIX_BUS_OFF_REC`
- `CONFIG_TWAI_ERRATA_FIX_RX_FIFO_CORRUPT`
- `CONFIG_TWAI_ERRATA_FIX_TX_INTR_LOST`
- `CONFIG_TWAI_ISR_IN_IRAM`

**CONFIG_TWAI_ISR_IN_IRAM**

Place TWAI ISR function into IRAM

*Found in: Component config > Driver Configurations > TWAI Configuration*

Place the TWAI ISR in to IRAM. This will allow the ISR to avoid cache misses, and also be able to run whilst the cache is disabled (such as when writing to SPI Flash). Note that if this option is enabled: - Users should also set the ESP_INTR_FLAG_IRAM in the driver configuration structure when installing the driver (see docs for specifics). - Alert logging (i.e., setting of the TWAI_ALERT_AND_LOG flag) will have no effect.

Default value:
- No (disabled)

**CONFIG_TWAI_ERRATA_FIX_BUS_OFF_REC**

Add SW workaround for REC change during bus-off

*Found in: Component config > Driver Configurations > TWAI Configuration*

When the bus-off condition is reached, the REC should be reset to 0 and frozen (via LOM) by the driver’s ISR. However on the ESP32, there is an edge case where the REC will increase before the driver’s ISR can respond in time (e.g., due to the rapid occurrence of bus errors), thus causing the REC to be non-zero after bus-off. A non-zero REC can prevent bus-off recovery as the bus-off recovery condition is that both TEC and REC become 0. Enabling this option will add a workaround in the driver to forcibly reset REC to zero on reaching bus-off.

Default value:
- Yes (enabled)

**CONFIG_TWAI_ERRATA_FIX_TX_INTR_LOST**

Add SW workaround for TX interrupt lost errata

*Found in: Component config > Driver Configurations > TWAI Configuration*

On the ESP32, when a transmit interrupt occurs, and interrupt register is read on the same APB clock cycle, the transmit interrupt could be lost. Enabling this option will add a workaround that checks the transmit buffer status bit to recover any lost transmit interrupt.

Default value:
- Yes (enabled)
CONFIG_TWAI_ERRATA_FIX_RX_FRAME_INVALID

Add SW workaround for invalid RX frame errata

*Found in: Component config > Driver Configurations > TWAI Configuration*

On the ESP32, when receiving a data or remote frame, if a bus error occurs in the data or CRC field, the data of the next received frame could be invalid. Enabling this option will add a workaround that will reset the peripheral on detection of this errata condition. Note that if a frame is transmitted on the bus whilst the reset is ongoing, the message will not be receive by the peripheral sent on the bus during the reset, the message will be lost.

**Default value:**
- Yes (enabled)

CONFIG_TWAI_ERRATA_FIX_RX_FIFO_CORRUPT

Add SW workaround for RX FIFO corruption errata

*Found in: Component config > Driver Configurations > TWAI Configuration*

On the ESP32, when the RX FIFO overruns and the RX message counter maxes out at 64 messages, the entire RX FIFO is no longer recoverable. Enabling this option will add a workaround that resets the peripheral on detection of this errata condition. Note that if a frame is being sent on the bus during the reset bus during the reset, the message will be lost.

**Default value:**
- Yes (enabled)

Temperature sensor Configuration

Contains:
- CONFIG_TEMP_SENSOR_ENABLE_DEBUG_LOG
- CONFIG_TEMP_SENSOR_SUPPRESS_DEPRECATE_WARN

CONFIG_TEMP_SENSOR_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > Temperature sensor Configuration*

Wether to suppress the deprecation warnings when using legacy temperature sensor driver (driver/temp_sensor.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
- No (disabled) if SOC_TEMP_SENSOR_SUPPORTED

CONFIG_TEMP_SENSOR_ENABLE_DEBUG_LOG

Enable debug log

*Found in: Component config > Driver Configurations > Temperature sensor Configuration*

Wether to enable the debug log message for temperature sensor driver. Note that, this option only controls the temperature sensor driver log, won’t affect other drivers.

**Default value:**
- No (disabled) if SOC_TEMP_SENSOR_SUPPORTED

UART Configuration

Contains:
- CONFIG_UART_ISR_IN_IRAM
CONFIG_UART_ISR_IN_IRAM

Place UART ISR function into IRAM

*Found in: Component config > Driver Configurations > UART Configuration*

If this option is not selected, UART interrupt will be disabled for a long time and may cause data lost when doing spi flash operation.

**Default value:**
- No (disabled) if `CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH`

GPIO Configuration  Contains:

- `CONFIG_GPIO_CTRL_FUNC_IN_IRAM`
- `CONFIG_GPIO_ESP32_SUPPORT_SWITCH_SLP_PULL`

CONFIG_GPIO_ESP32_SUPPORT_SWITCH_SLP_PULL

Support light sleep GPIO pullup/pulldown configuration for ESP32

*Found in: Component config > Driver Configurations > GPIO Configuration*

This option is intended to fix the bug that ESP32 is not able to switch to configured pullup/pulldown mode in sleep. If this option is selected, chip will automatically emulate the behaviour of switching, and about 450B of source codes would be placed into IRAM.

CONFIG_GPIO_CTRL_FUNC_IN_IRAM

Place GPIO control functions into IRAM

*Found in: Component config > Driver Configurations > GPIO Configuration*

Place GPIO control functions (like intr_disable/set_level) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context.

**Default value:**
- No (disabled)

Sigma Delta Modulator Configuration  Contains:

- `CONFIG_SDM_ENABLE_DEBUG_LOG`
- `CONFIG_SDM_CTRL_FUNC_IN_IRAM`
- `CONFIG_SDM_SUPPRESS_DEPRECATED_WARN`

CONFIG_SDM_CTRL_FUNC_IN_IRAM

Place SDM control functions into IRAM

*Found in: Component config > Driver Configurations > Sigma Delta Modulator Configuration*

Place SDM control functions (like set_duty) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

**Default value:**
- No (disabled)
**CONFIG_SDM_SUPPRESS_DEPRECATE_WARN**

Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > Sigma Delta Modulator Configuration*

Whether to suppress the deprecation warnings when using legacy sigma delta driver. If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
- No (disabled)

**CONFIG_SDM_ENABLE_DEBUG_LOG**

Enable debug log

*Found in: Component config > Driver Configurations > Sigma Delta Modulator Configuration*

Whether to enable the debug log message for SDM driver. Note that, this option only controls the SDM driver log, won’t affect other drivers.

**Default value:**
- No (disabled)

**GPTimer Configuration**

Contains:
- **CONFIG_GPTIMER_ENABLE_DEBUG_LOG**
- **CONFIG_GPTIMER_ISR_IRAM_SAFE**
- **CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM**
- **CONFIG_GPTIMER_SUPPRESS_DEPRECATE_WARN**

**CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM**

Place GPTimer control functions into IRAM

*Found in: Component config > Driver Configurations > GPTimer Configuration*

Place GPTimer control functions (like start/stop) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

**Default value:**
- No (disabled)

**CONFIG_GPTIMER_ISR_IRAM_SAFE**

GPTimer ISR IRAM-Safe

*Found in: Component config > Driver Configurations > GPTimer Configuration*

Ensure the GPTimer interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

**Default value:**
- No (disabled)

**CONFIG_GPTIMER_SUPPRESS_DEPRECATE_WARN**

Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > GPTimer Configuration*
Wether to suppress the deprecation warnings when using legacy timer group driver (driver/timer.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
- No (disabled)

**CONFIG_GPTIMER_ENABLE_DEBUG_LOG**
Enable debug log

*Found in: Component config > Driver Configurations > GPTimer Configuration*

Wether to enable the debug log message for GPTimer driver. Note that, this option only controls the GPTimer driver log, won’t affect other drivers.

**Default value:**
- No (disabled)

**PCNT Configuration**
Contains:
- `CONFIG_PCNT_ENABLE_DEBUG_LOG`
- `CONFIG_PCNT_ISR_IRAM.Safe`
- `CONFIG_PCNT_CTRL_FUNC_IN_IRAM`
- `CONFIG_PCNT_SUPPRESS_DEPRECATE_WARN`

**CONFIG_PCNT_CTRL_FUNC_IN_IRAM**
Place PCNT control functions into IRAM

*Found in: Component config > Driver Configurations > PCNT Configuration*

Place PCNT control functions (like start/stop) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

**Default value:**
- No (disabled)

**CONFIG_PCNT_ISR_IRAM_SAFE**
PCNT ISR IRAM-Safe

*Found in: Component config > Driver Configurations > PCNT Configuration*

Ensure the PCNT interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

**Default value:**
- No (disabled)

**CONFIG_PCNT_SUPPRESS_DEPRECATE_WARN**
Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > PCNT Configuration*

Wether to suppress the deprecation warnings when using legacy PCNT driver (driver/pcnt.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
- No (disabled)
CONFIG_PCNT_ENABLE_DEBUG_LOG

Enable debug log

*Found in: Component config > Driver Configurations > PCNT Configuration*

Whether to enable the debug log message for PCNT driver. Note that, this option only controls the PCNT driver log, won’t affect other drivers.

**Default value:**
- No (disabled)

**RMT Configuration** Contains:
- `CONFIG_RMT_ENABLE_DEBUG_LOG`
- `CONFIG_RMT_ISR_IRAM_SAFE`
- `CONFIG_RMT_SUPPRESS_DEPRECATE_WARN`

CONFIG_RMT_ISR_IRAM_SAFE

RMT ISR IRAM-Safe

*Found in: Component config > Driver Configurations > RMT Configuration*

Ensure the RMT interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

**Default value:**
- No (disabled)

CONFIG_RMT_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > RMT Configuration*

Whether to suppress the deprecation warnings when using legacy rmt driver (driver/rmt.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
- No (disabled)

CONFIG_RMT_ENABLE_DEBUG_LOG

Enable debug log

*Found in: Component config > Driver Configurations > RMT Configuration*

Whether to enable the debug log message for RMT driver. Note that, this option only controls the RMT driver log, won’t affect other drivers.

**Default value:**
- No (disabled)

MCPWM Configuration Contains:
- `CONFIG_MCPWM_ENABLE_DEBUG_LOG`
- `CONFIG_MCPWM_CTRL_FUNC_IN_IRAM`
- `CONFIG_MCPWM_ISR_IRAM_SAFE`
- `CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN`
CONFIG_MCPWM_ISR_IRAM_SAFE

Place MCPWM ISR function into IRAM

*Found in: Component config > Driver Configurations > MCPWM Configuration*

This will ensure the MCPWM interrupt handle is IRAM-Safe, allow to avoid flash cache misses, and also be able to run whilst the cache is disabled. (e.g. SPI Flash write)

**Default value:**
- No (disabled)

CONFIG_MCPWM_CTRL_FUNC_IN_IRAM

Place MCPWM control functions into IRAM

*Found in: Component config > Driver Configurations > MCPWM Configuration*

Place MCPWM control functions (like set_compare_value) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

**Default value:**
- No (disabled)

CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN

Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > MCPWM Configuration*

Whether to suppress the deprecation warnings when using legacy MCPWM driver (driver/mcpwm.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
- No (disabled)

CONFIG_MCPWM_ENABLE_DEBUG_LOG

Enable debug log

*Found in: Component config > Driver Configurations > MCPWM Configuration*

Whether to enable the debug log message for MCPWM driver. Note that, this option only controls the MCPWM driver log, won’t affect other drivers.

**Default value:**
- No (disabled)

**I2S Configuration**

Contains:
- CONFIG_I2S_ENABLE_DEBUG_LOG
- CONFIG_I2S_ISR_IRAM_SAFE
- CONFIG_I2S_SUPPRESS_DEPRECATE_WARN

CONFIG_I2S_ISR_IRAM_SAFE

I2S ISR IRAM-Safe

*Found in: Component config > Driver Configurations > I2S Configuration*

Ensure the I2S interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).
Default value:
  • No (disabled)

**CONFIG_I2S.Suppress_Deprecate.Warn**
Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > I2S Configuration*

Enable this option will suppress the deprecation warnings of using APIs in legacy I2S driver.

Default value:
  • No (disabled)

**CONFIG_I2S_Enable_Debug_Log**
Enable I2S debug log

*Found in: Component config > Driver Configurations > I2S Configuration*

Whether to enable the debug log message for I2S driver. Note that, this option only controls the I2S driver log, will not affect other drivers.

Default value:
  • No (disabled)

**DAC Configuration**
Contains:
  • CONFIG_DAC_DMA.Auto_16Bit.Align
  • CONFIG_DAC_ISR.IRAM.Safe
  • CONFIG_DAC_Enable_Debug_Log
  • CONFIG_DAC_CTRL.Func.In.IRAM
  • CONFIG_DAC.Suppress_Deprecate.Warn

**CONFIG_DAC_CTRL.Func.In.IRAM**
Place DAC control functions into IRAM

*Found in: Component config > Driver Configurations > DAC Configuration*

Place DAC control functions (e.g. `dac_oneshot_output_voltage`) into IRAM, so that this function can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

Default value:
  • No (disabled)

**CONFIG_DAC_ISR.IRAM.Safe**
DAC ISR IRAM-Safe

*Found in: Component config > Driver Configurations > DAC Configuration*

Ensure the DAC interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write).

Default value:
  • No (disabled)
CONFIG_DAC_SUPPRESS_DEPRECATED_WARN
Suppress legacy driver deprecated warning

*Found in: Component config > Driver Configurations > DAC Configuration*

Whether to suppress the deprecation warnings when using legacy DAC driver (driver/dac.h). If you want to continue using the legacy driver, and don’t want to see related deprecation warnings, you can enable this option.

**Default value:**
- No (disabled)

CONFIG_DAC_ENABLE_DEBUG_LOG
Enable debug log

*Found in: Component config > Driver Configurations > DAC Configuration*

Whether to enable the debug log message for DAC driver. Note that, this option only controls the DAC driver log, won’t affect other drivers.

**Default value:**
- No (disabled)

CONFIG_DAC_DMA_AUTO_16BIT_ALIGN
Align the continuous data to 16 bit automatically

*Found in: Component config > Driver Configurations > DAC Configuration*

Whether to left shift the continuous data to align every bytes to 16 bits in the driver. On ESP32, although the DAC resolution is only 8 bits, the hardware requires 16 bits data in continuous mode. By enabling this option, the driver will left shift 8 bits for the input data automatically. Only disable this option when you decide to do this step by yourself. Note that the driver will allocate a new piece of memory to save the converted data.

**Default value:**
- Yes (enabled)

eFuse Bit Manager Contains:
- CONFIG_EFUSE_CODE_SCHEME_SELECTOR
- CONFIG_EFUSE_VIRTUAL
- CONFIG_EFUSE_CUSTOM_TABLE

CONFIG_EFUSE_CUSTOM_TABLE
Use custom eFuse table

*Found in: Component config > eFuse Bit Manager*

Allows to generate a structure for eFuse from the CSV file.

**Default value:**
- No (disabled)

CONFIG_EFUSE_CUSTOM_TABLE_FILENAME
Custom eFuse CSV file

*Found in: Component config > eFuse Bit Manager > CONFIG_EFUSE_CUSTOM_TABLE*

Name of the custom eFuse CSV filename. This path is evaluated relative to the project root directory.
Chapter 2. API 参考

**Default value:**
- "main/esp_efuse_custom_table.csv" if CONFIG_EFUSE_CUSTOM_TABLE

**CONFIG_EFUSE_VIRTUAL**

Simulate eFuse operations in RAM

*Found in: Component config > eFuse Bit Manager*

If “n” - No virtual mode. All eFuse operations are real and use eFuse registers. If “y” - The virtual mode is enabled and all eFuse operations (read and write) are redirected to RAM instead of eFuse registers, all permanent changes (via eFuse) are disabled. Log output will state changes that would be applied, but they will not be.

During startup, the eFuses are copied into RAM. This mode is useful for fast tests.

**Default value:**
- No (disabled)

**CONFIG_EFUSE_VIRTUAL_KEEP_IN_FLASH**

Keep eFuses in flash

*Found in: Component config > eFuse Bit Manager > CONFIG_EFUSE_VIRTUAL*

In addition to the “Simulate eFuse operations in RAM” option, this option just adds a feature to keep eFuses after reboots in flash memory. To use this mode the partition_table should have the efuse partition.

`partition.csv`: `efuse_em, data, efuse, . 0x2000`

During startup, the eFuses are copied from flash or, in case if flash is empty, from real eFuse to RAM and then update flash. This mode is useful when need to keep changes after reboot (testing secure.boot and flash_encryption).

**CONFIG_EFUSE_CODE_SCHEME_SELECTOR**

Coding Scheme Compatibility

*Found in: Component config > eFuse Bit Manager*

Selector eFuse code scheme.

**Available options:**
- None Only (EFUSE_CODE_SCHEME_COMPAT_NONE)
- 3/4 and None (EFUSE_CODE_SCHEME_COMPAT_3_4)
- Repeat, 3/4 and None (common table does not support it) (EFUSE_CODE_SCHEME_COMPAT_REPEAT)

**ESP-TLS** Contains:
- CONFIG_ESP_TLS_INSECURE
- CONFIG_ESP_TLS_LIBRARY_CHOOSE
- CONFIG_ESP_TLS_CLIENT_SESSION_TICKETS
- CONFIG_ESP_DEBUG_WOLFSSL
- CONFIG_ESP_TLS_SERVER
- CONFIG_ESP_TLS_PSK_VERIFICATION
- CONFIG_ESP_WOLFSSL_SMALL_CERT_VERIFY
- CONFIG_ESP_TLS_USE_DS_PERIPHERAL
- CONFIG_ESP_TLS_USE_SECURE_ELEMENT
**CONFIG_ESP_TLS_LIBRARY_CHOOSE**

Choose SSL/TLS library for ESP-TLS (See help for more Info)

*Found in: Component config > ESP-TLS*

The ESP-TLS APIs support multiple backend TLS libraries. Currently mbedTLS and WolfSSL are supported. Different TLS libraries may support different features and have different resource usage. Consult the ESP-TLS documentation in ESP-IDF Programming guide for more details.

**Available options:**
- mbedTLS (ESP_TLS_USING_MBEDTLS)
- wolfSSL (License info in wolfSSL directory README) (ESP_TLS_USING_WOLFSSL)

**CONFIG_ESP_TLS_USE_SECURE_ELEMENT**

Use Secure Element (ATECC608A) with ESP-TLS

*Found in: Component config > ESP-TLS*

Enable use of Secure Element for ESP-TLS, this enables internal support for ATECC608A peripheral on ESP-WROOM32SE, which can be used for TLS connection.

**CONFIG_ESP_TLS_USE_DS_PERIPHERAL**

Use Digital Signature (DS) Peripheral with ESP-TLS

*Found in: Component config > ESP-TLS*

Enable use of the Digital Signature Peripheral for ESP-TLS. The DS peripheral can only be used when it is appropriately configured for TLS. Consult the ESP-TLS documentation in ESP-IDF Programming Guide for more details.

**Default value:**
- Yes (enabled) if ESP_TLS_USING_MBEDTLS && SOC_DIG_SIGN_SUPPORTED

**CONFIG_ESP_TLS_CLIENT_SESSION_TICKETS**

Enable client session tickets

*Found in: Component config > ESP-TLS*

Enable session ticket support as specified in RFC5077.

**CONFIG_ESP_TLS_SERVER**

Enable ESP-TLS Server

*Found in: Component config > ESP-TLS*

Enable support for creating server side SSL/TLS session, available for mbedTLS as well as wolfSSL TLS library.

**CONFIG_ESP_TLS_SERVER_SESSION_TICKETS**

Enable server session tickets

*Found in: Component config > ESP-TLS > CONFIG_ESP_TLS_SERVER*

Enable session ticket support as specified in RFC5077
**CONFIG_ESP_TLS_SERVER_SESSION_TICKET_TIMEOUT**

Server session ticket timeout in seconds

*Found in:*  
Component config > ESP-TLS > CONFIG_ESP_TLS_SERVER > CONFIG_ESP_TLS_SERVER_SESSION_TICKETS

Sets the session ticket timeout used in the TLS server.

*Default value:*  
• 86400 if CONFIG_ESP_TLS_SERVER_SESSION_TICKETS

**CONFIG_ESP_TLS_SERVER_CERT_SELECT_HOOK**

Certificate selection hook

*Found in:*  
Component config > ESP-TLS > CONFIG_ESP_TLS_SERVER

Ability to configure and use a certificate selection callback during server handshake, to select a certificate to present to the client based on the TLS extensions supplied in the client hello (alpn, sni, etc).

**CONFIG_ESP_TLS_SERVER_MIN_AUTH_MODE_OPTIONAL**

ESP-TLS Server: Set minimum Certificate Verification mode to Optional

*Found in:*  
Component config > ESP-TLS > CONFIG_ESP_TLS_SERVER

When this option is enabled, the peer (here, the client) certificate is checked by the server, however the handshake continues even if verification failed. By default, the peer certificate is not checked and ignored by the server.

mbedtls_ssl_get_verify_result() can be called after the handshake is complete to retrieve status of verification.

**CONFIG_ESP_TLS_PSK_VERIFICATION**

Enable PSK verification

*Found in:*  
Component config > ESP-TLS

Enable support for pre shared key ciphers, supported for both mbedTLS as well as wolfSSL TLS library.

**CONFIG_ESP_TLS_INSECURE**

Allow potentially insecure options

*Found in:*  
Component config > ESP-TLS

You can enable some potentially insecure options. These options should only be used for testing purposes. Only enable these options if you are very sure.

**CONFIG_ESP_TLS_SKIP_SERVER_CERT_VERIFY**

Skip server certificate verification by default (WARNING: ONLY FOR TESTING PURPOSE, READ HELP)

*Found in:*  
Component config > ESP-TLS > CONFIG_ESP_TLS_INSECURE

After enabling this option the esp-tls client will skip the server certificate verification by default. Note that this option will only modify the default behaviour of esp-tls client regarding server cert verification. The default behaviour should only be applicable when no other option regarding the server cert verification is opted in the esp-tls config (e.g. crt_bundle_attach, use_global_ca_store etc.). WARNING : Enabling this option comes with a potential risk of establishing a TLS connection with a server which has a fake identity, provided that the server certificate is not provided either through API or other mechanism like ca_store etc.
CONFIG_ESP_WOLFSSL_SMALL_CERT_VERIFY
Enable SMALL_CERT_VERIFY

*Found in: Component config > ESP-TLS*

Enables server verification with Intermediate CA cert, does not authenticate full chain of trust up to the root CA cert (After Enabling this option client only needs to have Intermediate CA certificate of the server to authenticate server, root CA cert is not necessary).

**Default value:**
- Yes (enabled) if ESP_TLS_USING_WOLFSSL

CONFIG_ESP_DEBUG_WOLFSSL
Enable debug logs for wolfSSL

*Found in: Component config > ESP-TLS*

Enable detailed debug prints for wolfSSL SSL library.

ADC and ADC Calibration Contains:
- ADC Calibration Configurations
- CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE
- CONFIG_ADC_DISABLE_DAC_OUTPUT
- CONFIG_ADC_ONESHOT_CTRL_FUNC_IN_IRAM

CONFIG_ADC_ONESHOT_CTRL_FUNC_IN_IRAM
Place ISR version ADC oneshot mode read function into IRAM

*Found in: Component config > ADC and ADC Calibration*

Place ISR version ADC oneshot mode read function into IRAM.

**Default value:**
- No (disabled)

CONFIG_ADC_CONTINUOUS_ISR_IRAM_SAFE
ADC continuous mode driver ISR IRAM-Safe

*Found in: Component config > ADC and ADC Calibration*

Ensure the ADC continuous mode ISR is IRAM-Safe. When enabled, the ISR handler will be available when the cache is disabled.

**Default value:**
- No (disabled)

ADC Calibration Configurations Contains:
- CONFIG_ADC_CALI_EFUSE_VREF_ENABLE
- CONFIG_ADC_CALI_LUT_ENABLE
- CONFIG_ADC_CALI_EFUSE_TP_ENABLE
CONFIG_ADC_CALI_EFUSE_TP_ENABLE

Use Two Point Values

*Found in: Component config > ADC and ADC Calibration > ADC Calibration Configurations*

Some ESP32s have Two Point calibration values burned into eFuse BLOCK3. This option will allow the ADC calibration component to characterize the ADC-Voltage curve using Two Point values if they are available.

**Default value:**
- Yes (enabled)

CONFIG_ADC_CALI_EFUSE_VREF_ENABLE

Use eFuse Vref

*Found in: Component config > ADC and ADC Calibration > ADC Calibration Configurations*

Some ESP32s have Vref burned into eFuse BLOCK0. This option will allow the ADC calibration component to characterize the ADC-Voltage curve using eFuse Vref if it is available.

**Default value:**
- Yes (enabled)

CONFIG_ADC_CALI_LUT_ENABLE

Use Lookup Tables

*Found in: Component config > ADC and ADC Calibration > ADC Calibration Configurations*

This option will allow the ADC calibration component to use Lookup Tables to correct for non-linear behavior in 11db attenuation. Other attenuations do not exhibit non-linear behavior hence will not be affected by this option.

**Default value:**
- Yes (enabled)

CONFIG_ADC_DISABLE_DAC_OUTPUT

Disable DAC when ADC2 is in use

*Found in: Component config > ADC and ADC Calibration*

By default, this is set. The ADC oneshot driver will disable the output of the corresponding DAC channels: ESP32: IO25 and IO26 ESP32S2: IO17 and IO18

Disable this option so as to measure the output of DAC by internal ADC, for test usage.

**Default value:**
- Yes (enabled)

Common ESP-related Contains:
- CONFIG_ESP_ERR_TO_NAME_LOOKUP

CONFIG_ESP_ERR_TO_NAME_LOOKUP

Enable lookup of error code strings

*Found in: Component config > Common ESP-related*

Functions esp_err_to_name() and esp_err_to_name_r() return string representations of error codes from a pre-generated lookup table. This option can be used to turn off the use of the look-up table in order
to save memory but this comes at the price of sacrificing distinguishable (meaningful) output string representations.

**Default value:**
- Yes (enabled)

**Ethernet** Contains:
- `CONFIG_ETH_TRANSMIT_MUTEX`
- `CONFIG_ETH_USE_ESP32_EMAC`
- `CONFIG_ETH_USE_OPENETH`
- `CONFIG_ETH_USE_SPI_ETHERNET`

**CONFIG_ETH_USE_ESP32_EMAC**
Support ESP32 internal EMAC controller

*Found in: Component config > Ethernet*

ESP32 integrates a 10/100M Ethernet MAC controller.

**Default value:**
- Yes (enabled)

Contains:
- `CONFIG_ETH_DMA_RX_BUFFER_NUM`
- `CONFIG_ETH_DMA_TX_BUFFER_NUM`
- `CONFIG_ETH_SOFT_FLOW_CONTROL`
- `CONFIG_ETH_DMA_BUFFER_SIZE`
- `CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0`
- `CONFIG_ETH_PHY_INTERFACE`
- `CONFIG_ETH_RMII_CLK_OUT_GPIO`
- `CONFIG_ETH_RMII_CLK_MODE`

**CONFIG_ETH_PHY_INTERFACE**
PHY interface

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

Select the communication interface between MAC and PHY chip.

**Available options:**
- Reduced Media Independent Interface (RMII) (ETH_PHY_INTERFACE_RMII)

**CONFIG_ETH_RMII_CLK_MODE**
RMII clock mode

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

Select external or internal RMII clock.

**Available options:**
- Input RMII clock from external (ETH_RMII_CLK_INPUT) MAC will get RMII clock from outside. Note that ESP32 only supports GPIO0 to input the RMII clock.
- Output RMII clock from internal (ETH_RMII_CLK_OUTPUT) ESP32 can generate RMII clock by internal APLL. This clock can be routed to the external PHY device. ESP32 supports to route the RMII clock to GPIO0/16/17.
**CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0**

Output RMII clock from GPIO0 (Experimental!)

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

GPIO0 can be set to output a pre-divided PLL clock (test only!). Enabling this option will configure GPIO0 to output a 50MHz clock. In fact this clock doesn’t have directly relationship with EMAC peripheral. Sometimes this clock won’t work well with your PHY chip. You might need to add some extra devices after GPIO0 (e.g. inverter). Note that outputting RMII clock on GPIO0 is an experimental practice. If you want the Ethernet to work with WiFi, don’t select GPIO0 output mode for stability.

**Default value:**
- No (disabled) if ETH_RMII_CLK_OUTPUT && CONFIG_ETH_USE_ESP32_EMAC

**CONFIG_ETH_RMII_CLK_OUT_GPIO**

RMII clock GPIO number

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

Set the GPIO number to output RMII Clock.

**Range:**
- from 16 to 17 if CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0 && ETH_RMII_CLK_OUTPUT && CONFIG_ETH_USE_ESP32_EMAC

**Default value:**
- 17 if CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0 && ETH_RMII_CLK_OUTPUT && CONFIG_ETH_USE_ESP32_EMAC

**CONFIG_ETH_DMA_BUFFER_SIZE**

Ethernet DMA buffer size (Byte)

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

Set the size of each buffer used by Ethernet MAC DMA.

**Range:**
- from 256 to 1600

**Default value:**
- 512

**CONFIG_ETH_DMA_RX_BUFFER_NUM**

Amount of Ethernet DMA Rx buffers

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

Number of DMA receive buffers. Each buffer’s size is ETH_DMA_BUFFER_SIZE. Larger number of buffers could increase throughput somehow.

**Range:**
- from 3 to 30

**Default value:**
- 10

**CONFIG_ETH_DMA_TX_BUFFER_NUM**

Amount of Ethernet DMA Tx buffers

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

Number of DMA transmit buffers. Each buffer’s size is ETH_DMA_BUFFER_SIZE. Larger number of buffers could increase throughput somehow.
Range:
  • from 3 to 30
Default value:
  • 10

CONFIG_ETH_SOFT_FLOW_CONTROL
Enable software flow control

*Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC*

Ethernet MAC engine on ESP32 doesn’t feature a flow control logic. The MAC driver can perform a software flow control if you enable this option. Note that, if the RX buffer number is small, enabling software flow control will cause obvious performance loss.

Default value:
  • No (disabled) if CONFIG_ETH_DMA_RX_BUFFER_NUM > 15 && CONFIG_ETH_USE_ESP32_EMAC

CONFIG_ETH_USE_SPI_ETHERNET
Support SPI to Ethernet Module

*Found in: Component config > Ethernet*

ESP-IDF can also support some SPI-Ethernet modules.

Default value:
  • Yes (enabled)

Contains:
  • CONFIG_ETH_SPI_ETHERNET_DM9051
  • CONFIG_ETH_SPI_ETHERNET_KSZ8851SNL
  • CONFIG_ETH_SPI_ETHERNET_W5500

CONFIG_ETH_SPI_ETHERNET_DM9051
Use DM9051

*Found in: Component config > Ethernet > CONFIG_ETH_USE_SPI_ETHERNET*

DM9051 is a fast Ethernet controller with an SPI interface. It’s also integrated with a 10/100M PHY and MAC. Select this to enable DM9051 driver.

CONFIG_ETH_SPI_ETHERNET_W5500
Use W5500 (MAC RAW)

*Found in: Component config > Ethernet > CONFIG_ETH_USE_SPI_ETHERNET*

W5500 is a HW TCP/IP embedded Ethernet controller. TCP/IP stack, 10/100 Ethernet MAC and PHY are embedded in a single chip. However the driver in ESP-IDF only enables the RAW MAC mode, making it compatible with the software TCP/IP stack. Say yes to enable W5500 driver.

CONFIG_ETH_SPI_ETHERNET_KSZ8851SNL
Use KSZ8851SNL

*Found in: Component config > Ethernet > CONFIG_ETH_USE_SPI_ETHERNET*

The KSZ8851SNL is a single-chip Fast Ethernet controller consisting of a 10/100 physical layer transceiver (PHY), a MAC, and a Serial Peripheral Interface (SPI). Select this to enable KSZ8851SNL driver.
**CONFIG_ETH_USE_OPENETH**

Support OpenCores Ethernet MAC (for use with QEMU)

*Found in: Component config > Ethernet*

OpenCores Ethernet MAC driver can be used when an ESP-IDF application is executed in QEMU. This driver is not supported when running on a real chip.

**Default value:**
- No (disabled)

Contains:
- CONFIG_ETH_OPENETH_DMA_RX_BUFFER_NUM
- CONFIG_ETH_OPENETH_DMA_TX_BUFFER_NUM

**CONFIG_ETH_OPENETH_DMA_RX_BUFFER_NUM**

Number of Ethernet DMA Rx buffers

*Found in: Component config > Ethernet > CONFIG_ETH_USE_OPENETH*

Number of DMA receive buffers, each buffer is 1600 bytes.

**Range:**
- from 1 to 64 if CONFIG_ETH_USE_OPENETH

**Default value:**
- 4 if CONFIG_ETH_USE_OPENETH

**CONFIG_ETH_OPENETH_DMA_TX_BUFFER_NUM**

Number of Ethernet DMA Tx buffers

*Found in: Component config > Ethernet > CONFIG_ETH_USE_OPENETH*

Number of DMA transmit buffers, each buffer is 1600 bytes.

**Range:**
- from 1 to 64 if CONFIG_ETH_USE_OPENETH

**Default value:**
- 1 if CONFIG_ETH_USE_OPENETH

**CONFIG_ETH_TRANSMIT_MUTEX**

Enable Transmit Mutex

*Found in: Component config > Ethernet*

Prevents multiple accesses when Ethernet interface is used as shared resource and multiple functionalities might try to access it at a time.

**Default value:**
- No (disabled)

**Event Loop Library**

Contains:
- CONFIG_ESP_EVENT_LOOP_PROFILING
- CONFIG_ESP_EVENT_POST_FROM_ISR
CONFIG_ESP_EVENT_LOOP_PROFILING
Enable event loop profiling

*Found in: Component config > Event Loop Library*

Enables collections of statistics in the event loop library such as the number of events posted to/received by an event loop, number of callbacks involved, number of events dropped to a full event loop queue, run time of event handlers, and number of times/run time of each event handler.

**Default value:**
- No (disabled)

CONFIG_ESP_EVENT_POST_FROM_ISR
Support posting events from ISRs

*Found in: Component config > Event Loop Library*

Enable posting events from interrupt handlers.

**Default value:**
- Yes (enabled)

CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR
Support posting events from ISRs placed in IRAM

*Found in: Component config > Event Loop Library > CONFIG_ESP_EVENT_POST_FROM_ISR*

Enable posting events from interrupt handlers placed in IRAM. Enabling this option places API functions esp_event_post and esp_event_post_to in IRAM.

**Default value:**
- Yes (enabled)

GDB Stub   Contains:
- **CONFIG_ESP_GDBSTUB_SUPPORT_TASKS**

CONFIG_ESP_GDBSTUB_SUPPORT_TASKS
Enable listing FreeRTOS tasks through GDB Stub

*Found in: Component config > GDB Stub*

If enabled, GDBStub can supply the list of FreeRTOS tasks to GDB. Thread list can be queried from GDB using `info threads` command. Note that if GDB task lists were corrupted, this feature may not work. If GDBStub fails, try disabling this feature.

CONFIG_ESP_GDBSTUB_MAX_TASKS
Maximum number of tasks supported by GDB Stub

*Found in: Component config > GDB Stub > CONFIG_ESP_GDBSTUB_SUPPORT_TASKS*

Set the number of tasks which GDB Stub will support.

**Default value:**
- 32 if **CONFIG_ESP_GDBSTUB_SUPPORT_TASKS**
Chapter 2. API

ESP HTTP client  Contains:

- `CONFIG_ESP_HTTP_CLIENT_ENABLE_BASIC_AUTH`
- `CONFIG_ESP_HTTP_CLIENT_ENABLE_DIGEST_AUTH`
- `CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS`

`CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS`

Enable https

*Found in: Component config > ESP HTTP client*

This option will enable https protocol by linking esp-tls library and initializing SSL transport

**Default value:**

- Yes (enabled)

`CONFIG_ESP_HTTP_CLIENT_ENABLE_BASIC_AUTH`

Enable HTTP Basic Authentication

*Found in: Component config > ESP HTTP client*

This option will enable HTTP Basic Authentication. It is disabled by default as Basic auth uses unencrypted encoding, so it introduces a vulnerability when not using TLS

**Default value:**

- No (disabled)

`CONFIG_ESP_HTTP_CLIENT_ENABLE_DIGEST_AUTH`

Enable HTTP Digest Authentication

*Found in: Component config > ESP HTTP client*

This option will enable HTTP Digest Authentication. It is enabled by default, but use of this configuration is not recommended as the password can be derived from the exchange, so it introduces a vulnerability when not using TLS

**Default value:**

- No (disabled)

HTTP Server  Contains:

- `CONFIG_HTTPD_QUEUE_WORK_BLOCKING`
- `CONFIG_HTTPD_PURGE_BUF_LEN`
- `CONFIG_HTTPD_LOG_PURGE_DATA`
- `CONFIG_HTTPD_MAX_REQ_HDR_LEN`
- `CONFIG_HTTPD_MAX_URI_LEN`
- `CONFIG_HTTPD_ERR_RESP_NO_DELAY`
- `CONFIG_HTTPD_WS_SUPPORT`

`CONFIG_HTTPD_MAX_REQ_HDR_LEN`

Max HTTP Request Header Length

*Found in: Component config > HTTP Server*

This sets the maximum supported size of headers section in HTTP request packet to be processed by the server

**Default value:**

- 512
Chapter 2. API

**CONFIG_HTTPD_MAX_URI_LEN**

Max HTTP URI Length

*Found in: Component config > HTTP Server*

This sets the maximum supported size of HTTP request URI to be processed by the server

**Default value:**
- 512

**CONFIG_HTTPD_ERR_RESP_NO_DELAY**

Use TCP_NODELAY socket option when sending HTTP error responses

*Found in: Component config > HTTP Server*

Using TCP_NODEALY socket option ensures that HTTP error response reaches the client before the underlying socket is closed. Please note that turning this off may cause multiple test failures

**Default value:**
- Yes (enabled)

**CONFIG_HTTPD_PURGE_BUF_LEN**

Length of temporary buffer for purging data

*Found in: Component config > HTTP Server*

This sets the size of the temporary buffer used to receive and discard any remaining data that is received from the HTTP client in the request, but not processed as part of the server HTTP request handler.

If the remaining data is larger than the available buffer size, the buffer will be filled in multiple iterations. The buffer should be small enough to fit on the stack, but large enough to avoid excessive iterations.

**Default value:**
- 32

**CONFIG_HTTPD_LOG_PURGE_DATA**

Log purged content data at Debug level

*Found in: Component config > HTTP Server*

Enabling this will log discarded binary HTTP request data at Debug level. For large content data this may not be desirable as it will clutter the log.

**Default value:**
- No (disabled)

**CONFIG_HTTPD_WS_SUPPORT**

WebSocket server support

*Found in: Component config > HTTP Server*

This sets the WebSocket server support.

**Default value:**
- No (disabled)
CONFIG_HTTPD_QUEUE_WORK_BLOCKING

httpd_queue_work as blocking API

*Found in: Component config > HTTP Server*

This makes httpd_queue_work() API to wait until a message space is available on UDP control socket. It internally uses a counting semaphore with count set to LWIP_UDP_RECVMBOX_SIZE to achieve this. This config will slightly change API behavior to block until message gets delivered on control socket.

ESP HTTPS OTA Contains:

- CONFIG_ESP_HTTPS_OTA_ALLOW_HTTP
- CONFIG_ESP_HTTPS_OTA_DECRYPT_CB

CONFIG_ESP_HTTPS_OTA_DECRYPT_CB

Provide decryption callback

*Found in: Component config > ESP HTTPS OTA*

Exposes an additional callback whereby firmware data could be decrypted before being processed by OTA update component. This can help to integrate external encryption related format and removal of such encapsulation layer from firmware image.

**Default value:**
- No (disabled)

CONFIG_ESP_HTTPS_OTA.Allow_HTTP

Allow HTTP for OTA (WARNING: ONLY FOR TESTING PURPOSE, READ HELP)

*Found in: Component config > ESP HTTPS OTA*

It is highly recommended to keep HTTPS (along with server certificate validation) enabled. Enabling this option comes with potential risk of: - Non-encrypted communication channel with server - Accepting firmware upgrade image from server with fake identity

**Default value:**
- No (disabled)

ESP HTTPS server Contains:

- CONFIG_ESP_HTTPS_SERVER_ENABLE

CONFIG_ESP_HTTPS_SERVER_ENABLE

Enable ESP_HTTPS_SERVER component

*Found in: Component config > ESP HTTPS server*

Enable ESP HTTPS server component

Hardware Settings Contains:

- Chip revision
- ETM Configuration
- GDMA Configuration
- MAC Config
- Main XTAL Config
- MMU Config
- Peripheral Control
Chip revision  
**Contains:**

- `CONFIG_ESP32_REV_MIN`

**CONFIG_ESP32_REV_MIN**

Minimum Supported ESP32 Revision

*Found in: Component config > Hardware Settings > Chip revision*

Required minimum chip revision. ESP-IDF will check for it and reject to boot if the chip revision fails the check. This ensures the chip used will have some modifications (features, or bugfixes).

The compiled binary will only support chips above this revision, this will also help to reduce binary size.

**Available options:**

- Rev v0.0 (ECO0) (`ESP32_REV_MIN_0`)
- Rev v1.0 (ECO1) (`ESP32_REV_MIN_1`)
- Rev v1.1 (ECO1.1) (`ESP32_REV_MIN_1.1`)
- Rev v2.0 (ECO2) (`ESP32_REV_MIN_2`)
- Rev v3.0 (ECO3) (`ESP32_REV_MIN_3`)
- Rev v3.1 (ECO4) (`ESP32_REV_MIN_3.1`)

**MAC Config**  
**Contains:**

- `CONFIG_ESP32_UNIVERSAL_MAC_ADDRESSES`

**CONFIG_ESP32_UNIVERSAL_MAC_ADDRESSES**

Number of universally administered (by IEEE) MAC address

*Found in: Component config > Hardware Settings > MAC Config*

Configure the number of universally administered (by IEEE) MAC addresses. During initialization, MAC addresses for each network interface are generated or derived from a single base MAC address. If the number of universal MAC addresses is four, all four interfaces (WiFi station, WiFi softap, Bluetooth and Ethernet) receive a universally administered MAC address. These are generated sequentially by adding 0, 1, 2 and 3 (respectively) to the final octet of the base MAC address. If the number of universal MAC addresses is two, only two interfaces (WiFi station and Bluetooth) receive a universally administered MAC address. These are generated sequentially by adding 0 and 1 (respectively) to the base MAC address. The remaining two interfaces (WiFi softap and Ethernet) receive local MAC addresses. These are derived from the universal WiFi station and Bluetooth MAC addresses, respectively. When using the default (Espressif-assigned) base MAC address, either setting can be used. When using a custom universal MAC address range, the correct setting will depend on the allocation of MAC addresses in this range (either 2 or 4 per device.)

**Available options:**

- Two (`ESP32_UNIVERSAL_MAC_ADDRESSES_TWO`)
- Four (`ESP32_UNIVERSAL_MAC_ADDRESSES_FOUR`)

**Sleep Config**  
**Contains:**

- `CONFIG_ESP_SLEEP_MSPI_NEED_ALL_IO_PU`
- `CONFIG_ESP_SLEEP_DEEP_SLEEP_WAKEUP_DELAY`
- `CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND`
- `CONFIG_ESP_SLEEP_GPIO_RESET_WORKAROUND`
- `CONFIG_ESP_SLEEP_POWER_DOWN_FLASH`
- `CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND`
**CONFIG_ESP_SLEEP_POWER_DOWN_FLASH**

Power down flash in light sleep when there is no SPIRAM

*Found in: Component config > Hardware Settings > Sleep Config*

If enabled, chip will try to power down flash as part of esp_light_sleep_start(), which costs more time when chip wakes up. Can only be enabled if there is no SPIRAM configured.

This option will power down flash under a strict but relatively safe condition. Also, it is possible to power down flash under a relaxed condition by using esp_sleep_pd_config() to set ESP_PD_DOMAIN_VDDSDIO to ESP_PD_OPTION_OFF. It should be noted that there is a risk in powering down flash, you can refer ESP-IDF Programming Guide/API Reference/System API/Sleep Modes/Power-down of Flash for more details.

**Default value:**

- No (disabled) if CONFIG_SPIRAM

**CONFIG_ESP_SLEEP_GPIO_RESET_WORKAROUND**

Light sleep GPIO reset workaround

*Found in: Component config > Hardware Settings > Sleep Config*

esp32c2, esp32c3 and esp32s3 will reset at wake-up if GPIO is received a small electrostatic pulse during light sleep, with specific condition

- GPIO needs to be configured as input-mode only
- The pin receives a small electrostatic pulse, and reset occurs when the pulse voltage is higher than 6 V

For GPIO set to input mode only, it is not a good practice to leave it open/floating. The hardware design needs to controlled it with determined supply or ground voltage is necessary.

This option provides a software workaround for this issue. Configure to isolate all GPIO pins in sleep state.

**CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND**

PSRAM leakage current workaround in light sleep

*Found in: Component config > Hardware Settings > Sleep Config*

When the CS pin of SPIRAM is not pulled up, the sleep current will increase during light sleep. If the CS pin of SPIRAM has an external pull-up, you do not need to select this option, otherwise, you should enable this option.

**Default value:**

- Yes (enabled) if CONFIG_SPIRAM

**CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND**

Flash leakage current workaround in light sleep

*Found in: Component config > Hardware Settings > Sleep Config*

When the CS pin of Flash is not pulled up, the sleep current will increase during light sleep. If the CS pin of Flash has an external pull-up, you do not need to select this option, otherwise, you should enable this option.

**Default value:**

- Yes (enabled)
CONFIG_ESP_SLEEP_MSPI_NEED_ALL_IO PU

All pins of mspi need pull up

*Found in: Component config > Hardware Settings > Sleep Config*

To reduce leakage current, some types of SPI Flash/RAM only need to pull up the CS pin during light sleep. But there are also some kinds of SPI Flash/RAM that need to pull up all pins. It depends on the SPI Flash/RAM chip used.

CONFIG_ESP_SLEEP_DEEP_SLEEP_WAKEUP_DELAY

Extra delay in deep sleep wake stub (in us)

*Found in: Component config > Hardware Settings > Sleep Config*

When the chip exits deep sleep, the CPU and the flash chip are powered on at the same time. CPU will run deep sleep stub first, and then proceed to load code from flash. Some flash chips need sufficient time to pass between power on and first read operation. By default, without any extra delay, this time is approximately 900us, although some flash chip types need more than that.

By default extra delay is set to 2000us. When optimizing startup time for applications which require it, this value may be reduced.

If you are seeing “flash read err, 1000” message printed to the console after deep sleep reset, try increasing this value.

*Range:*
  - from 0 to 5000

*Default value:*
  - 2000

RTC Clock Config  Contains:

- CONFIG_RTC_EXT_CRYST_ADDIT_CURRENT_METHOD
- CONFIG_RTC_CLOCK_BBPLL_POWER_ON_WITH_USB
- CONFIG_RTC_XTAL_CAL_RETRY
- CONFIG_RTC_CLK_CAL_CYCLES
- CONFIG_RTC_CLK_SRC

CONFIG_RTC_CLK_SRC

RTC clock source

*Found in: Component config > Hardware Settings > RTC Clock Config*

Choose which clock is used as RTC clock source.

- **“Internal 150kHz oscillator” option provides lowest deep sleep current** consumption, and does not require extra external components. However frequency stability with respect to temperature is poor, so time may drift in deep/light sleep modes.
- **“External 32kHz crystal” provides better frequency stability, at the expense of slightly higher (1uA) deep sleep current consumption.**
- **“External 32kHz oscillator” allows using 32kHz clock generated by an external circuit.** In this case, external clock signal must be connected to 32K_XN pin. Amplitude should be <1.2V in case of sine wave signal, and <1V in case of square wave signal. Common mode voltage should be 0.1 < Vcm < 0.5Vamp, where Vamp is the signal amplitude. Additionally, 1nF capacitor must be connected between 32K_XP pin and ground. 32K_XP pin cannot be used as a GPIO in this case.
- **“Internal 8.5MHz oscillator divided by 256” option results in higher deep sleep current** (by 5uA) but has better frequency stability than the internal 150kHz oscillator. It does not require external components.
Available options:

- Internal 150kHz RC oscillator (RTC_CLK_SRC_INT_RC)
- External 32kHz crystal (RTC_CLK_SRC_EXT_CRYS)
- External 32kHz oscillator at 32K_XN pin (RTC_CLK_SRC_EXT_OSC)
- Internal 8.5MHz oscillator, divided by 256 (~33kHz) (RTC_CLK_SRC_INT_8MD256)

**CONFIG_RTC_EXT_CRYST_ADDIT_CURRENT_METHOD**

Additional current for external 32kHz crystal

*Found in: Component config > Hardware Settings > RTC Clock Config*

With some 32kHz crystal configurations, the X32N and X32P pins may not have enough drive strength to keep the crystal oscillating. Choose the method to provide additional current from touchpad 9 to the external 32kHz crystal. Note that the deep sleep current is slightly high (4-5uA) and the touchpad and the wakeup sources of both touchpad and ULP are not available in method 1 and method 2.

This problem is fixed in ESP32 ECO 3, so this workaround is not needed. Setting the project configuration to minimum revision ECO3 will disable this option, allow all wakeup sources, and save some code size.

- “None” option will not provide additional current to external crystal
- “Method 1” option cannot ensure 100% to solve the external 32k crystal start failed issue, but the touchpad can work in this method.
- “Method 2” option can solve the external 32k issue, but the touchpad cannot work in this method.

Available options:

- None (RTC_EXT_CRYST_ADDIT_CURRENT_NONE)
- Method 1 (RTC_EXT_CRYST_ADDIT_CURRENT)
- Method 2 (RTC_EXT_CRYST_ADDIT_CURRENT_V2)

**CONFIG_RTC_CLK_CAL_CYCLES**

Number of cycles for RTC_SLOW_CLK calibration

*Found in: Component config > Hardware Settings > RTC Clock Config*

When the startup code initializes RTC_SLOW_CLK, it can perform calibration by comparing the RTC_SLOW_CLK frequency with main XTAL frequency. This option sets the number of RTC_SLOW_CLK cycles measured by the calibration routine. Higher numbers increase calibration precision, which may be important for applications which spend a lot of time in deep sleep. Lower numbers reduce startup time.

When this option is set to 0, clock calibration will not be performed at startup, and approximate clock frequencies will be assumed:

- 150000 Hz if internal RC oscillator is used as clock source. For this use value 1024.
- 32768 Hz if the 32k crystal oscillator is used. For this use value 3000 or more. In case more value will help improve the definition of the launch of the crystal. If the crystal could not start, it will be switched to internal RC.

**Range:**

- from 0 to 27000 if RTC_CLK_SRC_EXT_CRYS \(\lor\) RTC_CLK_SRC_EXT_OSC \(\lor\) RTC_CLK_SRC_INT_8MD256
- from 0 to 32766

**Default value:**

- 3000 if RTC_CLK_SRC_EXT_CRYS \(\lor\) RTC_CLK_SRC_EXT_OSC \(\lor\) RTC_CLK_SRC_INT_8MD256
- 1024
**CONFIG_RTC_XTAL_CAL_RETRY**

Number of attempts to repeat 32k XTAL calibration

---

**Found in:** Component config > Hardware Settings > RTC Clock Config

---

Number of attempts to repeat 32k XTAL calibration before giving up and switching to the internal RC. Increase this option if the 32k crystal oscillator does not start and switches to internal RC.

---

**Default value:**

- 1 if RTC_CLK_SRC_EXT_CRYSTAL

**CONFIG_RTC_CLOCK_BBPLL_POWER_ON_WITH_USB**

Keep BBPLL clock always work

---

**Found in:** Component config > Hardware Settings > RTC Clock Config

---

When software switches the CPU clock source from BBPLL clock to XTAL, usually the BBPLL will be switched off. This helps to save some power consumption in sleep modes. However this may also happen during the software reset, resulting in the inactive (disconnected from host) of the USB_SERIAL_JTAG device during software reset.

When USB_SERIAL_JTAG is being used, whether to turn off the clock source during software reset and in sleep modes is determined by RTC_CLOCK_BBPLL_POWER_ON_WITH_USB.

- When RTC_CLOCK_BBPLL_POWER_ON_WITH_USB is enabled, the clock will be kept, so that the USB_SERIAL_JTAG will keep alive during software reset. The side-effect is the increasing of power consumption during sleep modes, even though USB_SERIAL_JTAG will not work in sleep modes.
- When RTC_CLOCK_BBPLL_POWER_ON_WITH_USB is disabled, the clock will be turned off. USB_SERIAL_JTAG will be inactive during software reset and in sleep modes. This saves some power consumption in sleep modes.

When USB_SERIAL_JTAG is not being used, software will always turn off BBPLL regardless of RTC_CLOCK_BBPLL_POWER_ON_WITH_USB is set or not.

**Default value:**

- Yes (enabled) if ESP_CONSOLE_USB_SERIAL_JTAG || ESP_CONSOLE_SECONDARY_USB_SERIAL_JTAG

**Peripheral Control**

Contains:

- **CONFIG_PERIPH_CTRL_FUNC_IN_IRAM**

**CONFIG_PERIPH_CTRL_FUNC_IN_IRAM**

Place peripheral control functions into IRAM

---

**Found in:** Component config > Hardware Settings > Peripheral Control

---

Place peripheral control functions (e.g. periph_module_reset) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context.

**Default value:**

- No (disabled)

**ETM Configuration**

Contains:

- **CONFIG_ETM_ENABLE_DEBUG_LOG**
CONFIG_ETM_ENABLE_DEBUG_LOG

Enable debug log

*Found in: Component config > Hardware Settings > ETM Configuration*

Whether to enable the debug log message for ETM core driver. Note that, this option only controls the ETM related driver log, won’t affect other drivers.

**Default value:**
- No (disabled) if SOC_ETM_SUPPORTED

MMU Config

GDMA Configuration  Contains:
- CONFIG_GDMA_ISR_IRAM_SAFE
- CONFIG_GDMA_CTRL_FUNC_IN_IRAM

**CONFIG_GDMA_CTRL_FUNC_IN_IRAM**

Place GDMA control functions into IRAM

*Found in: Component config > Hardware Settings > GDMA Configuration*

Place GDMA control functions (like start/stop/append/reset) into IRAM, so that these functions can be IRAM-safe and able to be called in the other IRAM interrupt context. Enabling this option can improve driver performance as well.

**Default value:**
- No (disabled) if SOC_GDMA_SUPPORTED

**CONFIG_GDMA_ISR_IRAM_SAFE**

GDMA ISR IRAM-Safe

*Found in: Component config > Hardware Settings > GDMA Configuration*

This will ensure the GDMA interrupt handler is IRAM-Safe, allow to avoid flash cache misses, and also be able to run whilst the cache is disabled. (e.g. SPI Flash write).

**Default value:**
- No (disabled) if SOC_GDMA_SUPPORTED

Main XTAL Config  Contains:
- CONFIG_XTAL_FREQ_SEL

**CONFIG_XTAL_FREQ_SEL**

Main XTAL frequency

*Found in: Component config > Hardware Settings > Main XTAL Config*

This option selects the operating frequency of the XTAL (crystal) clock used to drive the ESP target. The selected value MUST reflect the frequency of the given hardware.

Note: The XTAL_FREQ_AUTO option allows the ESP target to automatically estimating XTAL clock’s operating frequency. However, this feature is only supported on the ESP32. The ESP32 uses the internal 8MHZ as a reference when estimating. Due to the internal oscillator’s frequency being temperature dependent, usage of the XTAL_FREQ_AUTO is not recommended in applications that operate in high ambient temperatures or use high-temperature qualified chips and modules.
Available options:
  • 24 MHz (XTAL_FREQ_24)
  • 26 MHz (XTAL_FREQ_26)
  • 32 MHz (XTAL_FREQ_32)
  • 40 MHz (XTAL_FREQ_40)
  • Autodetect (XTAL_FREQ_AUTO)

LCD and Touch Panel  Contains:
  • LCD Peripheral Configuration

LCD Peripheral Configuration  Contains:
  • CONFIG_LCD_ENABLE_DEBUG_LOG
  • CONFIG_LCD_PANEL_IO_FORMAT_BUF_SIZE
  • CONFIG_LCD_RGB_RESTART_IN_VSYNC
  • CONFIG_LCD_RGB_ISR_IRAM_SAFE

CONFIG_LCD_PANEL_IO_FORMAT_BUF_SIZE
LCD panel io format buffer size

*Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration*

LCD driver allocates an internal buffer to transform the data into a proper format, because of the endian order mismatch. This option is to set the size of the buffer, in bytes.

Default value:
  • 32

CONFIG_LCD_ENABLE_DEBUG_LOG
Enable debug log

*Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration*

Whether to enable the debug log message for LCD driver. Note that, this option only controls the LCD driver log, won’t affect other drivers.

Default value:
  • No (disabled)

CONFIG_LCD_RGB_ISR_IRAM_SAFE
RGB LCD ISR IRAM-Safe

*Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration*

Ensure the LCD interrupt is IRAM-Safe by allowing the interrupt handler to be executable when the cache is disabled (e.g. SPI Flash write). If you want the LCD driver to keep flushing the screen even when cache ops disabled, you can enable this option. Note, this will also increase the IRAM usage.

Default value:
  • No (disabled) if SOC_LCD_RGB_SUPPORTED

CONFIG_LCD_RGB_RESTART_IN_VSYNC
Restart transmission in VSYNC

*Found in: Component config > LCD and Touch Panel > LCD Peripheral Configuration*

Reset the GDMA channel every VBlank to stop permanent desyncs from happening. Only need to enable it when in your application, the DMA can’t deliver data as fast as the LCD consumes it.
**Default value:**
- No (disabled) if SOC_LCD_RGB_SUPPORTED

**ESP NETIF Adapter**

Contains:

- `CONFIG_ESP_NETIF_BRIDGE_EN`
- `CONFIG_ESP_NETIF_L2_TAP`
- `CONFIG_ESP_NETIF_IP_LOST_TIMER_INTERVAL`
- `CONFIG_ESP_NETIF_USE_TCPIP_STACK_LIB`

**CONFIG_ESP_NETIF_IP_LOST_TIMER_INTERVAL**

IP Address lost timer interval (seconds)

*Found in: Component config > ESP NETIF Adapter*

The value of 0 indicates the IP lost timer is disabled, otherwise the timer is enabled.

The IP address may be lost because of some reasons, e.g. when the station disconnects from soft-AP, or when DHCP IP renew fails etc. If the IP lost timer is enabled, it will be started everytime the IP is lost. Event `SYSTEM_EVENT_STA_LOST_IP` will be raised if the timer expires. The IP lost timer is stopped if the station get the IP again before the timer expires.

*Range:*
- from 0 to 65535

*Default value:*
- 120

**CONFIG_ESP_NETIF_USE_TCPIP_STACK_LIB**

TCP/IP Stack Library

*Found in: Component config > ESP NETIF Adapter*

Choose the TCP/IP Stack to work, for example, LwIP, uIP, etc.

*Available options:*

- LwIP (ESP_NETIF_TCPIP_LWIP)
  - lwIP is a small independent implementation of the TCP/IP protocol suite.
- Loopback (ESP_NETIF_LOOPBACK)
  - Dummy implementation of esp-netif functionality which connects driver transmit to receive function. This option is for testing purpose only

**CONFIG_ESP_NETIF_L2_TAP**

Enable netif L2 TAP support

*Found in: Component config > ESP NETIF Adapter*

A user program can read/write link layer (L2) frames from/to ESP TAP device. The ESP TAP device can be currently associated only with Ethernet physical interfaces.

**CONFIG_ESP_NETIF_L2_TAP_MAX_FDS**

Maximum number of opened L2 TAP File descriptors

*Found in: Component config > ESP NETIF Adapter > CONFIG_ESP_NETIF_L2_TAP*

Maximum number of opened File descriptors (FD’s) associated with ESP TAP device. ESP TAP FD’s take up a certain amount of memory, and allowing fewer FD’s to be opened at the same time conserves memory.

*Range:*
• from 1 to 10 if CONFIG_ESP_NETIF_L2_TAP

Default value:
• 5 if CONFIG_ESP_NETIF_L2_TAP

CONFIG_ESP_NETIF_L2_TAP_RX_QUEUE_SIZE

Size of L2 TAP Rx queue

Found in: Component config > ESP NETIF Adapter > CONFIG_ESP_NETIF_L2_TAP

Maximum number of frames queued in opened File descriptor. Once the queue is full, the newly arriving frames are dropped until the queue has enough room to accept incoming traffic (Tail Drop queue management).

Range:
• from 1 to 100 if CONFIG_ESP_NETIF_L2_TAP

Default value:
• 20 if CONFIG_ESP_NETIF_L2_TAP

CONFIG_ESP_NETIF_BRIDGE_EN

Enable LwIP IEEE 802.1D bridge

Found in: Component config > ESP NETIF Adapter

Enable LwIP IEEE 802.1D bridge support in ESP-NETIF. Note that “Number of clients store data in netif” (LWIP_NUM_NETIF_CLIENT_DATA) option needs to be properly configured to be LwIP bridge available!

Default value:
• No (disabled)

PHY

Contains:

• CONFIG_ESP_PHY_ENABLE_USB
• CONFIG_ESP_PHY_MAX_WIFI_TX_POWER
• CONFIG_ESP_PHY_REDUCE_TX_POWER
• CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE
• CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION

CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE

Store phy calibration data in NVS

Found in: Component config > PHY

If this option is enabled, NVS will be initialized and calibration data will be loaded from there. PHY calibration will be skipped on deep sleep wakeup. If calibration data is not found, full calibration will be performed and stored in NVS. Normally, only partial calibration will be performed. If this option is disabled, full calibration will be performed.

If it’s easy that your board calibrate bad data, choose ‘n’. Two cases for example, you should choose ‘n’: 1. If your board is easy to be booted up with antenna disconnected. 2. Because of your board design, each time when you do calibration, the result are too unstable. If unsure, choose ‘y’.

Default value:
• Yes (enabled)
Chapter 2. API

**CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION**

Use a partition to store PHY init data

*Found in: Component config > PHY*

If enabled, PHY init data will be loaded from a partition. When using a custom partition table, make sure that PHY data partition is included (type: ‘data’, subtype: ‘phy’). With default partition tables, this is done automatically. If PHY init data is stored in a partition, it has to be flashed there, otherwise runtime error will occur.

If this option is not enabled, PHY init data will be embedded into the application binary.

If unsure, choose ‘n’.

**Default value:**

- No (disabled)

**Contains:**

- **CONFIG_ESP_PHY_DEFAULT_INIT_IF_INVALID**
- **CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN**

**CONFIG_ESP_PHY_DEFAULT_INIT_IF_INVALID**

Reset default PHY init data if invalid

*Found in: Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*

If enabled, PHY init data will be restored to default if it cannot be verified successfully to avoid endless bootloops.

If unsure, choose ‘n’.

**Default value:**

- No (disabled) if **CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION**

**CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN**

Support multiple PHY init data bin

*Found in: Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION*

If enabled, the corresponding PHY init data type can be automatically switched according to the country code. China’s PHY init data bin is used by default. Can be modified by country information in API esp_wifi_set_country(). The priority of switching the PHY init data type is: 1. Country configured by API esp_wifi_set_country() and the parameter policy is WIFI_COUNTRY_POLICY_MANUAL. 2. Country notified by the connected AP. 3. Country configured by API esp_wifi_set_country() and the parameter policy is WIFI_COUNTRY_POLICY_AUTO.

**Default value:**

- No (disabled) if **CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION** && **CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN**

**CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN_EMBED**

Support embedded multiple phy init data bin to app bin

*Found in: Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION > CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN*

If enabled, multiple phy init data bin will embedded into app bin. If not enabled, multiple phy init data bin will still leave alone, and need to be flashed by users.

**Default value:**

- No (disabled) if **CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN** && **CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION**
**CONFIG_ESP_PHY_INIT_DATA_ERROR**

Terminate operation when PHY init data error

*Found in:* Component config > PHY > CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION > CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN

If enabled, when an error occurs while the PHY init data is updated, the program will terminate and restart. If not enabled, the PHY init data will not be updated when an error occurs.

**Default value:**
- No (disabled) if CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN && CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION

**CONFIG_ESP_PHY_MAX_WIFI_TX_POWER**

Max WiFi TX power (dBm)

*Found in:* Component config > PHY

Set maximum transmit power for WiFi radio. Actual transmit power for high data rates may be lower than this setting.

**Range:**
- from 10 to 20

**Default value:**
- 20

**CONFIG_ESP_PHY_REDUCE_TX_POWER**

Reduce PHY TX power when brownout reset

*Found in:* Component config > PHY

When brownout reset occurs, reduce PHY TX power to keep the code running.

**Default value:**
- Yes (enabled)

**CONFIG_ESP_PHY_ENABLE_USB**

Enable USB when phy init

*Found in:* Component config > PHY

When using USB Serial/JTAG/OTG/CDC, PHY should enable USB, otherwise USB module can not work properly. Notice: Enabling this configuration option will slightly impact wifi performance.

**Default value:**
- No (disabled) if ESP_CONSOLE_USB_SERIAL_JTAG || ESP_CONSOLE_SECONDARY_USB_SERIAL_JTAG

**Power Management**

Contains:

- CONFIG_PM_SLP_DISABLE_GPIO
- CONFIG_PM_SLP_IRAM_OPT
- CONFIG_PMRTOS_IDLE_OPT
- CONFIG_PM_ENABLE
**CONFIG_PM_ENABLE**

Support for power management

*Found in: Component config > Power Management*

If enabled, application is compiled with support for power management. This option has run-time overhead (increased interrupt latency, longer time to enter idle state), and it also reduces accuracy of RTOS ticks and timers used for timekeeping. Enable this option if application uses power management APIs.

**Default value:**
- No (disabled) if `CONFIG_FREERTOS_SMP` || `__DOXYGEN__`

**CONFIG_PM_DFS_INIT_AUTO**

Enable dynamic frequency scaling (DFS) at startup

*Found in: Component config > Power Management > CONFIG_PM_ENABLE*

If enabled, startup code configures dynamic frequency scaling. Max CPU frequency is set to DEFAULT_CPU_FREQ_MHZ setting, min frequency is set to XTAL frequency. If disabled, DFS will not be active until the application configures it using esp_pm_configure function.

**Default value:**
- No (disabled) if `CONFIG_PM_ENABLE`

**CONFIG_PM_PROFILING**

Enable profiling counters for PM locks

*Found in: Component config > Power Management > CONFIG_PM_ENABLE*

If enabled, esp_pm_* functions will keep track of the amount of time each of the power management locks has been held, and esp_pm_dump_locks function will print this information. This feature can be used to analyze which locks are preventing the chip from going into a lower power state, and see what time the chip spends in each power saving mode. This feature does incur some run-time overhead, so should typically be disabled in production builds.

**Default value:**
- No (disabled) if `CONFIG_PM_ENABLE`

**CONFIG_PM_TRACE**

Enable debug tracing of PM using GPIOs

*Found in: Component config > Power Management > CONFIG_PM_ENABLE*

If enabled, some GPIOs will be used to signal events such as RTOS ticks, frequency switching, entry/exit from idle state. Refer to pm_trace.c file for the list of GPIOs. This feature is intended to be used when analyzing/debugging behavior of power management implementation, and should be kept disabled in applications.

**Default value:**
- No (disabled) if `CONFIG_PM_ENABLE`

**CONFIG_PM_SLP_IRAM_OPT**

Put lightsleep related codes in internal RAM

*Found in: Component config > Power Management*

If enabled, about 1.8KB of lightsleep related source code would be in IRAM and chip would sleep longer for 760us at most each time. This feature is intended to be used when lower power consumption is needed while there is enough place in IRAM to place source code.
**CONFIG_PM_RTOS_IDLE_OPT**

Put RTOS IDLE related codes in internal RAM

*Found in: Component config > Power Management*

If enabled, about 260B of RTOS IDLE related source code would be in IRAM and chip would sleep longer for 40us at most each time. This feature is intended to be used when lower power consumption is needed while there is enough place in IRAM to place source code.

**CONFIG_PM_SLP_DISABLE_GPIO**

Disable all GPIO when chip at sleep

*Found in: Component config > Power Management*

This feature is intended to disable all GPIO pins at automatic sleep to get a lower power mode. If enabled, chips will disable all GPIO pins at automatic sleep to reduce about 200~300 uA current. If you want to specifically use some pins normally as chip wakes when chip sleeps, you can call `gpio_sleep_sel_disable` to disable this feature on those pins. You can also keep this feature on and call `gpio_sleep_set_direction` and `gpio_sleep_set_pull_mode` to have a different GPIO configuration at sleep. Warning: If you want to enable this option on ESP32, you should enable `GPIO_ESP32_SUPPORT_SWITCH_SLP_PULL` at first, otherwise you will not be able to switch pullup/pulldown mode.

**ESP PSRAM**

Contains:

- `CONFIG_SPIRAM`

**CONFIG_SPIRAM**

Support for external, SPI-connected RAM

*Found in: Component config > ESP PSRAM*

This enables support for an external SPI RAM chip, connected in parallel with the main SPI flash chip.

**Default value:**

- No (disabled)

**SPI RAM config**

Contains:

- `CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY`
- `CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY`
- `CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY`
- `CONFIG_SPIRAM_SPIWP_SD3_PIN`
- `CONFIG_SPIRAM_BANKSWITCH_ENABLE`
- `CONFIG_SPIRAM_2T_MODE`
- `CONFIG_SPIRAM_CACHE_WORKAROUND`
- `CONFIG_SPIRAM_BOOT_INIT`
- `CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL`
- `CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL`
- `CONFIG_SPIRAM_MEMTEST`
- `CONFIG_SPIRAM_SPEED`
- `CONFIG_SPIRAM_OCCUPY_SPI_HOST`
- `CONFIG_SPIRAM_USE`
- SPIRAM cache workaround debugging
- SPIRAM workaround libraries placement
Chapter 2. API

- `CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP`
- `CONFIG_SPIRAM_TYPE`
- `CONFIG_SPIRAM_CUSTOM_SPIWP_SD3_PIN`

**CONFIG_SPIRAM_TYPE**

Type of SPI RAM chip in use

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

**Available options:**
- Auto-detect (SPIRAM_TYPE_AUTO)
- ESP-PSRAM16 or APS1604 (SPIRAM_TYPE_ESPPSRAM16)
- ESP-PSRAM32 (SPIRAM_TYPE_ESPPSRAM32)
- ESP-PSRAM64 or LY68L6400 (SPIRAM_TYPE_ESPPSRAM64)

**CONFIG_SPIRAM_SPEED**

Set RAM clock speed

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

Select the speed for the SPI RAM chip. If SPI RAM is enabled, we only support three combinations of SPI speed mode we supported now:

1. Flash SPI running at 40Mhz and RAM SPI running at 40Mhz
2. Flash SPI running at 80Mhz and RAM SPI running at 40Mhz
3. Flash SPI running at 80Mhz and RAM SPI running at 80Mhz

*Note: If the third mode (80Mhz+80Mhz) is enabled for SPI RAM of type 32MBit, one of the HSPI/VSPI host will be occupied by the system. Which SPI host to use can be selected by the config item SPIRAM_OCCUPY_SPI_HOST. Application code should never touch HSPI/VSPI hardware in this case. The option to select 80Mhz will only be visible if the flash SPI speed is also 80Mhz. (ESP_TOOLPY_FLASHFREQ_80M is true)*

**Available options:**
- 40MHz clock speed (SPIRAM_SPEED_40M)
- 80MHz clock speed (SPIRAM_SPEED_80M)

**CONFIG_SPIRAM_BOOT_INIT**

Initialize SPI RAM during startup

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

If this is enabled, the SPI RAM will be enabled during initial boot. Unless you have specific requirements, you’ll want to leave this enabled so memory allocated during boot-up can also be placed in SPI RAM.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM`

**CONFIG_SPIRAM_IGNORE_NOTFOUND**

Ignore PSRAM when not found

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > CONFIG_SPIRAM_BOOT_INIT*

Normally, if psram initialization is enabled during compile time but not found at runtime, it is seen as an error making the CPU panic. If this is enabled, booting will complete but no PSRAM will be available.

**Default value:**
- No (disabled) if `CONFIG_SPIRAM_BOOT_INIT` & `CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY` & `CONFIG_SPIRAM`
CONFIG_SPIRAM_USE

SPI RAM access method

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

The SPI RAM can be accessed in multiple methods: by just having it available as an unmanaged memory region in the CPU’s memory map, by integrating it in the heap as ‘special’ memory needing heap_caps_malloc to allocate, or by fully integrating it making malloc() also able to return SPI RAM pointers.

*Available options:*
- Integrate RAM into memory map (SPIRAM_USE_MEMMAP)
- Make RAM allocatable using heap_caps_malloc(…, MALLOC_CAP_SPIRAM) (SPIRAM_USE_CAPS_ALLOC)
- Make RAM allocatable using malloc() as well (SPIRAM_USE_MALLOC)

CONFIG_SPIRAM_MEMTEST

Run memory test on SPI RAM initialization

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

Runs a rudimentary memory test on initialization. aborts when memory test fails. Disable this for slightly faster startup.

*Default value:*
- Yes (enabled) if CONFIG_SPIRAM_BOOT_INIT && CONFIG_SPIRAM

CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL

Maximum malloc() size, in bytes, to always put in internal memory

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

If malloc() is capable of also allocating SPI-connected ram, its allocation strategy will prefer to allocate chunks less than this size in internal memory, while allocations larger than this will be done from external RAM. If allocation from the preferred region fails, an attempt is made to allocate from the non-preferred region instead, so malloc() will not suddenly fail when either internal or external memory is full.

*Range:*
- from 0 to 131072 if SPIRAM_USE_MALLOC && CONFIG_SPIRAM

*Default value:*
- 16384 if SPIRAM_USE_MALLOC && CONFIG_SPIRAM

CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP

Try to allocate memories of WiFi and LWIP in SPIRAM firstly. If failed, allocate internal memory

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

Try to allocate memories of WiFi and LWIP in SPIRAM firstly. If failed, try to allocate internal memory then.

*Default value:*
- No (disabled) if (SPIRAM_USE_CAPS_ALLOC || SPIRAM_USE_MALLOC) && CONFIG_SPIRAM

CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL

Reserve this amount of bytes for data that specifically needs to be in DMA or internal memory

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*
Because the external/internal RAM allocation strategy is not always perfect, it sometimes may happen that the internal memory is entirely filled up. This causes allocations that are specifically done in internal memory, for example the stack for new tasks or memory to service DMA or have memory that’s also available when SPI cache is down, to fail. This option reserves a pool specifically for requests like that; the memory in this pool is not given out when a normal malloc() is called.

Set this to 0 to disable this feature.

Note that because FreeRTOS stacks are forced to internal memory, they will also use this memory pool; be sure to keep this in mind when adjusting this value.

Note also that the DMA reserved pool may not be one single contiguous memory region, depending on the configured size and the static memory usage of the app.

Range:
- from 0 to 262144 if SPIRAM_USE_MALLOC && CONFIG_SPIRAM

Default value:
- 32768 if SPIRAM_USE_MALLOC && CONFIG_SPIRAM

**CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY**

Allow .bss segment placed in external memory

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

If enabled, variables with EXT_RAM_BSS_ATTR attribute will be placed in SPIRAM instead of internal DRAM. BSS section of lwip, net80211, pp, bt libraries will be automatically placed in SPIRAM. BSS sections from other object files and libraries can also be placed in SPIRAM through linker fragment scheme *extram_bss*.

Note that the variables placed in SPIRAM using EXT_RAM_BSS_ATTR will be zero initialized.

Default value:
- No (disabled) if CONFIG_SPIRAM && CONFIG_SPIRAM

**CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY**

Allow .noinit segment placed in external memory

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

If enabled, noinit variables can be placed in PSRAM using EXT_RAM_NOINIT_ATTR.

Note the values placed into this section will not be initialized at startup and should keep its value after software restart.

Default value:
- No (disabled) if CONFIG_SPIRAM && CONFIG_SPIRAM

**CONFIG_SPIRAM_CACHE_WORKAROUND**

Enable workaround for bug in SPI RAM cache for Rev1 ESP32s

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

Revision 1 of the ESP32 has a bug that can cause a write to PSRAM not to take place in some situations when the cache line needs to be fetched from external RAM and an interrupt occurs. This enables a fix in the compiler (-mfix-esp32-psram-cache-issue) that makes sure the specific code that is vulnerable to this will not be emitted.

This will also not use any bits of newlib that are located in ROM, opting for a version that is compiled with the workaround and located in flash instead.

The workaround is not required for ESP32 revision 3 and above.

Default value:
Chapter 2. API

SPIRAM cache workaround debugging

Contains:

- CONFIG_SPIRAM_CACHE_WORKAROUND_STRATEGY

CONFIG_SPIRAM_CACHE_WORKAROUND_STRATEGY

Workaround strategy

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM cache workaround debugging

Select the workaround strategy. Note that the strategy for precompiled libraries (libgcc, newlib, bt, wifi) is not affected by this selection.

Unless you know you need a different strategy, it’s suggested you stay with the default MEMW strategy. Note that DUPLDST can interfere with hardware encryption and this will be automatically disabled if this workaround is selected. ‘Insert nops’ is the workaround that was used in older esp-idf versions. This workaround still can cause faulty data transfers from/to SPI RAM in some situation.

Available options:

- Insert memw after vulnerable instructions (default) (SPIRAM_CACHE_WORKAROUND_STRATEGY_MEMW)
- Duplicate LD/ST for 32-bit, memw for 8/16 bit (SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST)
- Insert nops between vulnerable loads/stores (old strategy, obsolete) (SPIRAM_CACHE_WORKAROUND_STRATEGY_NOPS)

SPIRAM workaround libraries placement

Contains:

- CONFIG_SPIRAM_CACHE_LIBCHAR_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBENV_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBFILE_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBIO_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBJMP_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBMATH_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBMEM_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBMISC_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBNUMPARSER_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBRAND_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBSTR_IN_IRAM
- CONFIG_SPIRAM_CACHE_LIBTIME_IN_IRAM

CONFIG_SPIRAM_CACHE_LIBJMP_IN_IRAM

Put libc’s jump related functions in IRAM

Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement

The functions affected by this option are: longjmp and setjmp. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

Default value:

- Yes (enabled) if CONFIG_SPIRAM_CACHE_WORKAROUND && CONFIG_SPIRAM
**CONFIG_SPIRAM_CACHE_LIBMATH_IN_IRAM**

Put libc’s math related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: abs, div, labs, ldiv, quorem, fpclassify, and nan. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM_CACHE_LIBNUMPARSER_IN_IRAM**

Put libc’s number parsing related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: utoa, itoa, atoi, atol, strtol, and strtoul. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM_CACHE_LIBIO_IN_IRAM**

Put libc’s I/O related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: wcrtomb, ffwritee, wbuf, wsetup, fputwc, wcrtomb_r, ungetc, makebuf, fflush, refill, and sccl. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM_CACHE_LIBTIME_IN_IRAM**

Put libc’s time related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: asctime, asctime_r, ctime, ctime_r, lctime, lctime_r, gmtime, gmtime_r, strftime, mktime, tzset, tzset_r, tzset, time, gettzinfo, systimes, month_lengths, timelocal, tzvart, tzlock, tzcalc_limits, and strftime. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM_CACHE_LIBCHAR_IN_IRAM**

Put libc’s characters related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*
The functions affected by this option are: ctype, toupper, tolower, toascii,strupr, bzero, isalnum, isalpha, isascii, isblank, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, and isupper. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM CACHE_LIBMEM_IN_IRAM**

Put libc’s memory related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: memccpy, memchr, memmove, and memrchr. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM CACHE_LIBSTR_IN_IRAM**

Put libc’s string related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: strcasecmp, strcasestr, strchr, strcoll, strcpy, strcspn, strdup, strdup_r, strlcat, strlcpy, strlen, strlwr, strncasecmp, strncat, strncmp, strncpy, strndup, strndup_r, strsep, strspn, strstr, strtok_r, and strupr. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM CACHE_LIBRAND_IN_IRAM**

Put libc’s random related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: srand, rand, and rand_r. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM CACHE_LIBENV_IN_IRAM**

Put libc’s environment related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: environ, envlock, and getenv_r. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

**Default value:**
- Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND` && `CONFIG_SPIRAM`
• Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND && CONFIG_SPIRAM`

**CONFIG_SPIRAM_CACHE_LIBFILE_IN_IRAM**

Put libc’s file related functions in IRAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: lock, isatty, fclose, open, close, creat, read, rshift, sbrk, stdio, systm, syscall, sysopen, creat, sysread, syswrite, impure, fwalk, and findfp. Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

*Default value:*

• Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND && CONFIG_SPIRAM`

**CONFIG_SPIRAM_CACHE_LIBMISC_IN_IRAM**

Put libc’s miscellaneous functions in IRAM, see help

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > SPIRAM workaround libraries placement*

The functions affected by this option are: raise and system Putting these function in IRAM will allow them to be called when flash cache is disabled but it will also reduce the available size of free IRAM for the user application.

*Default value:*

• Yes (enabled) if `CONFIG_SPIRAM_CACHE_WORKAROUND && CONFIG_SPIRAM`

**CONFIG_SPIRAM_BANKSWITCH_ENABLE**

Enable bank switching for >4MiB external RAM

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

The ESP32 only supports 4MiB of external RAM in its address space. The hardware does support larger memories, but these have to be bank-switched in and out of this address space. Enabling this allows you to reserve some MMU pages for this, which allows the use of the esp_himem api to manage these banks.

#Note that this is limited to 62 banks, as esp_psram_extram_writeback_cache needs some kind of mapping of #some banks below that mark to work. We cannot at this moment guarantee this to exist when himem is #enabled.

If spiram 2T mode is enabled, the size of 64Mbit psram will be changed as 32Mbit, so himem will be unusable.

*Default value:*

• Yes (enabled) if `(SPIRAM_USE_MEMMAP || SPIRAM_USE_CAPS_ALLOC || SPIRAM_USE_MALLOC) && CONFIG_SPIRAM`

**CONFIG_SPIRAM_BANKSWITCH_RESERVE**

Amount of 32K pages to reserve for bank switching

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > CONFIG_SPIRAM_BANKSWITCH_ENABLE*

Select the amount of banks reserved for bank switching. Note that the amount of RAM allocatable with malloc/esp_heap_alloc_caps will decrease by 32K for each page reserved here.
Note that this reservation is only actually done if your program actually uses the himem API. Without any himem calls, the reservation is not done and the original amount of memory will be available to malloc/esp_heap_alloc_caps.

**Range:**
- from 1 to 62 if `CONFIG_SPIRAM_BANKSWITCH_ENABLE` && `CONFIG_SPIRAM`  

**Default value:**  
- 8 if `CONFIG_SPIRAM_BANKSWITCH_ENABLE` && `CONFIG_SPIRAM`  

**CONFIG_SPIRAMALLOW_STACK_EXTERNAL_MEMORY**

Allow external memory as an argument to xTaskCreateStatic

*Found in:* Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config

Because some bits of the ESP32 code environment cannot be recompiled with the cache workaround, normally tasks cannot be safely run with their stack residing in external memory; for this reason xTaskCreate (and related task creation functions) always allocate stack in internal memory and xTaskCreateStatic will check if the memory passed to it is in internal memory. If you have a task that needs a large amount of stack and does not call on ROM code in any way (no direct calls, but also no Bluetooth/WiFi), you can try enable this to cause xTaskCreateStatic to allow tasks stack in external memory.

**Default value:**  
- No (disabled) if SPIRAM_USE_MALLOC && `CONFIG_SPIRAM`  

**CONFIG_SPIRAM_OCCUPY_SPI_HOST**

SPI host to use for 32MBit PSRAM

*Found in:* Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config

When both flash and PSRAM is working under 80MHz, and the PSRAM is of type 32MBit, one of the HSPI/VSPI host will be used to output the clock. Select which one to use here.

**Available options:**  
- HSPI host (SPI2) (SPIRAM_OCCUPY_HSPI_HOST)  
- VSPI host (SPI3) (SPIRAM_OCCUPY_VSPI_HOST)  
- Will not try to use any host, will abort if not able to use the PSRAM (SPIRAM_OCCUPY_NO_HOST)

**PSRAM clock and cs IO for ESP32-DOWD**

Contains:

- `CONFIG_D0WD_PSRAM_CLK_IO`  
- `CONFIG_D0WD_PSRAM_CS_IO`

**CONFIG_D0WD_PSRAM_CLK_IO**

PSRAM CLK IO number

*Found in:* Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > PSRAM clock and cs IO for ESP32-DOWD

The PSRAM CLOCK IO can be any unused GPIO, user can config it based on hardware design. If user use 1.8V flash and 1.8V psram, this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

**Range:**
- from 0 to 33 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`  

**Default value:**  
- 17 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`
Chapter 2. API

**CONFIG_D0WD_PSRAM_CS_IO**

PSRAM CS IO number

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > PSRAM clock and cs IO for ESP32-D0WD*

The PSRAM CS IO can be any unused GPIO, user can config it based on hardware design. If user use 1.8V flash and 1.8V psram, this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

*Range:*
  - from 0 to 33 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

*Default value:*
  - 16 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

**PSRAM clock and cs IO for ESP32-D2WD**

Contains:

- `CONFIG_D2WD_PSRAM_CLK_IO`
- `CONFIG_D2WD_PSRAM_CS_IO`

**CONFIG_D2WD_PSRAM_CLK_IO**

PSRAM CLK IO number

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > PSRAM clock and cs IO for ESP32-D2WD*

User can config it based on hardware design. For ESP32-D2WD chip, the psram can only be 1.8V psram, so this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

*Range:*
  - from 0 to 33 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

*Default value:*
  - 9 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

**CONFIG_D2WD_PSRAM_CS_IO**

PSRAM CS IO number

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > PSRAM clock and cs IO for ESP32-D2WD*

User can config it based on hardware design. For ESP32-D2WD chip, the psram can only be 1.8V psram, so this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

*Range:*
  - from 0 to 33 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

*Default value:*
  - 10 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

**PSRAM clock and cs IO for ESP32-PICO**

Contains:

- `CONFIG_PICO_PSRAM_CS_IO`

**CONFIG_PICO_PSRAM_CS_IO**

PSRAM CS IO number

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config > PSRAM clock and cs IO for ESP32-PICO*

The PSRAM CS IO can be any unused GPIO, user can config it based on hardware design.
For ESP32-PICO chip, the psram share clock with flash, so user do not need to configure the clock IO. For the reference hardware design, please refer to https://www.espressif.com/sites/default/files/documentation/esp32-pico-d4_datasheet_en.pdf

**Range:**
- from 0 to 33 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

**Default value:**
- 10 if `CONFIG_SPIRAM` && `CONFIG_SPIRAM`

**CONFIG_SPIRAM_CUSTOM_SPIWP_SD3_PIN**

Use custom SPI PSRAM WP(SD3) Pin when flash pins set in eFuse (read help)

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

This setting is only used if the SPI flash pins have been overridden by setting the eFuses SPI_PAD_CONFIG_xxx, and the SPI flash mode is DIO or DOUT.

When this is the case, the eFuse config only defines 3 of the 4 Quad I/O data pins. The WP pin (aka ESP32 pin “SD_DATA_3” or SPI flash pin “IO2”) is not specified in eFuse. The psram only has QPI mode, so a WP pin setting is necessary.

If this config item is set to N (default), the correct WP pin will be automatically used for any Espressif chip or module with integrated flash. If a custom setting is needed, set this config item to Y and specify the GPIO number connected to the WP pin.

When flash mode is set to QIO or QOUT, the PSRAM WP pin will be set the same as the SPI Flash WP pin configured in the bootloader.

**Default value:**
- No (disabled) if (ESPTOOLPY_FLASHMODE_DIO || ESPTOOLPY_FLASHMODE_DOUT) && `CONFIG_SPIRAM`

**CONFIG_SPIRAM_SPIWP_SD3_PIN**

Custom SPI PSRAM WP(SD3) Pin

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

The option “Use custom SPI PSRAM WP(SD3) pin” must be set or this value is ignored.

If burning a customized set of SPI flash pins in eFuse and using DIO or DOUT mode for flash, set this value to the GPIO number of the SPIRAM WP pin.

**Range:**
- from 0 to 33 if (ESPTOOLPY_FLASHMODE_DIO || ESPTOOLPY_FLASHMODE_DOUT) && `CONFIG_SPIRAM`

**Default value:**
- 7 if (ESPTOOLPY_FLASHMODE_DIO || ESPTOOLPY_FLASHMODE_DOUT) && `CONFIG_SPIRAM`

**CONFIG_SPIRAM_2T_MODE**

Enable SPI PSRAM 2T mode

*Found in: Component config > ESP PSRAM > CONFIG_SPIRAM > SPI RAM config*

Enable this option to fix single bit errors inside 64Mbit PSRAM.

Some 64Mbit PSRAM chips have a hardware issue in the RAM which causes bit errors at multiple fixed bit positions.

Note: If this option is enabled, the 64Mbit PSRAM chip will appear to be 32Mbit in size. Applications will not be affected unless the use the esp_himem APIs, which are not supported in 2T mode.
### Default value:
- No (disabled) if `CONFIG_SPIRAM` & `CONFIG_SPIRAM` 

#### ESP Ringbuf
Contains:
- `CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH`
- `CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH`

#### `CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH`
Place non-ISR ringbuf functions into flash

*Found in: Component config > ESP Ringbuf*

Place non-ISR ringbuf functions (like `xRingbufferCreate/xRingbufferSend`) into flash. This frees up IRAM, but the functions can no longer be called when the cache is disabled.

**Default value:**
- No (disabled)

#### `CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH`
Place ISR ringbuf functions into flash

*Found in: Component config > ESP Ringbuf*

Place non-ISR ringbuf functions (like `xRingbufferSendFromISR/xRingbufferReceiveFromISR`) into flash. This frees up IRAM, but the functions can no longer be called when the cache is disabled or from an IRAM interrupt context.

This option is not compatible with ESP-IDF drivers which is configured to run the ISR from an IRAM context, e.g. `CONFIG_UART_ISR_IN_IRAM`.

**Default value:**
- No (disabled)

### ESP System Settings
Contains:
- `CONFIG_ESP_SYSTEM_RTC_EXT_XTAL_BOOTSTRAP_CYCLES`
- Brownout Detector
- `CONFIG_ESP_CONSOLE_UART`
- `CONFIG_ESP_CONSOLE_SECONDARY`
- `CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ`
- `CONFIG_ESP_SYSTEM_ALLOW_RTC_FAST_MEM_AS_HEAP`
- `CONFIG_ESP_TASK_WDT_EN`
- `CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE`
- `CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL`
- `CONFIG_ESP_INT_WDT`
- `CONFIG_ESP_MAIN_TASK_AFFINITY`
- `CONFIG_ESP_MAIN_TASK_STACK_SIZE`
- `CONFIG_ESP_DEBUG_OCDAWARE`
- Memory
- Memory protection
- `CONFIG_ESP_MINIMAL_SHARED_STACK_SIZE`
- `CONFIG_ESP_DEBUG_STUBS_ENABLE`
- `CONFIG_ESP_SYSTEM_PANIC`
- `CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE`
- `CONFIG_ESP_PANIC_HANDLER_IRAM`
- `CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE`
- Trace memory
- `CONFIG_ESP_CONSOLE_UART_BAUDRATE`
Chapter 2. API

- CONFIG_ESP_CONSOL Lease UART_NUM
- CONFIG_ESP_CONSOL Lease UART RX_GPIO
- CONFIG_ESP_CONSOL Lease UART TX_GPIO

CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ

CPU frequency

Found in: Component config > ESP System Settings

CPU frequency to be set on application startup.

Available options:
- 40 MHz (ESP_DEFAULT_CPU_FREQ_MHZ_40)
- 80 MHz (ESP_DEFAULT_CPU_FREQ_MHZ_80)
- 160 MHz (ESP_DEFAULT_CPU_FREQ_MHZ_160)
- 240 MHz (ESP_DEFAULT_CPU_FREQ_MHZ_240)

Memory Contains:

- CONFIG_ESP32_IRAM AS 8BIT ACCESSIBLE MEMORY
- CONFIG_ESP32_RTCDATA IN FAST_MEM
- CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE

CONFIG_ESP32_RTCDATA IN FAST_MEM

Place RTC_DATA_ATTR and RTC_RODATA_ATTR variables into RTC fast memory segment

Found in: Component config > ESP System Settings > Memory

This option allows to place .rtc_data and .rtc_rodata sections into RTC fast memory segment to free the slow memory region for ULP programs. This option depends on the CONFIG_FREERTOS_UNICORE option because RTC fast memory can be accessed only by PRO_CPU core.

Default value:
- No (disabled) if CONFIG_FREERTOS_UNICORE

CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE

Use fixed static RAM size

Found in: Component config > ESP System Settings > Memory

If this option is disabled, the DRAM part of the heap starts right after the .bss section, within the dram0_0 region. As a result, adding or removing some static variables will change the available heap size.

If this option is enabled, the DRAM part of the heap starts right after the dram0_0 region, where its length is set with ESP32_FIXED_STATIC_RAM_SIZE

Default value:
- No (disabled)

CONFIG_ESP32_FIXED_STATIC_RAM_SIZE

Fixed Static RAM size

Found in: Component config > ESP System Settings > Memory > CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE

RAM size dedicated for static variables (.data & .bss sections). Please note that the actual length will be reduced by BTDM_RESERVE_DRAM if Bluetooth controller is enabled.

Range:
• from 0 to 0x2c200 if CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE

**Default value:**
• “0x1E000” if CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE

**CONFIG_ESP32_IRAM_AS_8BIT_ACCESSIBLE_MEMORY**

Enable IRAM as 8 bit accessible memory

*Found in: Component config > ESP System Settings > Memory*

If enabled, application can use IRAM as byte accessible region for storing data (Note: IRAM region cannot be used as task stack)

This is possible due to handling of exceptions LoadStoreError (3) and LoadStoreAlignmentError (9) Each unaligned read/write access will incur a penalty of maximum of 167 CPU cycles.

**Trace memory**  
Contains:
• CONFIG_ESP32_TRAX

**CONFIG_ESP32_TRAX**

Use TRAX tracing feature

*Found in: Component config > ESP System Settings > Trace memory*

The ESP32 contains a feature which allows you to trace the execution path the processor has taken through the program. This is stored in a chunk of 32K (16K for single-processor) of memory that can’t be used for general purposes anymore. Disable this if you do not know what this is.

**Default value:**
• No (disabled)

**CONFIG_ESP32_TRAX_TWOBANKS**

Reserve memory for tracing both pro as well as app cpu execution

*Found in: Component config > ESP System Settings > Trace memory > CONFIG_ESP32_TRAX*

The ESP32 contains a feature which allows you to trace the execution path the processor has taken through the program. This is stored in a chunk of 32K (16K for single-processor) of memory that can’t be used for general purposes anymore. Disable this if you do not know what this is.

# Memory to reverse for trace, used in linker script

**Default value:**
• No (disabled) if CONFIG_ESP32_TRAX && CONFIG_FREERTOS_UNICORE

**CONFIG_ESP_SYSTEM_PANIC**

Panic handler behaviour

*Found in: Component config > ESP System Settings*

If FreeRTOS detects unexpected behaviour or an unhandled exception, the panic handler is invoked. Configure the panic handler’s action here.

**Available options:**
• Print registers and halt (ESP_SYSTEM_PANIC_PRINT_HALT)  
  Outputs the relevant registers over the serial port and halt the processor. Needs a manual reset to restart.
• Print registers and reboot (ESP_SYSTEM_PANIC_PRINT_REBOOT)  
  Outputs the relevant registers over the serial port and immediately reset the processor.
• Silent reboot (ESP_SYSTEM_PANIC_SILENT_REBOOT)
  Just resets the processor without outputting anything
• GDBStub on panic (ESP_SYSTEM_PANIC_GDBSTUB)
  Invoke gdbstub on the serial port, allowing for gdb to attach to it to do a postmortem of the crash.
• GDBStub at runtime (ESP_SYSTEM_GDBSTUB_RUNTIME)
  Invoke gdbstub on the serial port, allowing for gdb to attach to it and to do a debug on runtime.

CONFIG_ESP_SYSTEM_RTC_EXT_XTAL_BOOTSTRAP_CYCLES
Bootstrap cycles for external 32kHz crystal

*Found in: Component config > ESP System Settings*

To reduce the startup time of an external RTC crystal, we bootstrap it with a 32kHz square wave for a fixed number of cycles. Setting 0 will disable bootstrapping (if disabled, the crystal may take longer to start up or fail to oscillate under some conditions).

If this value is too high, a faulty crystal may initially start and then fail. If this value is too low, an otherwise good crystal may not start.

To accurately determine if the crystal has started, set a larger “Number of cycles for RTC_SLOW_CLK calibration” (about 3000).

CONFIG_ESP_SYSTEM_ALLOW_RTC_FAST_MEM_AS_HEAP
Enable RTC fast memory for dynamic allocations

*Found in: Component config > ESP System Settings*

This config option allows to add RTC fast memory region to system heap with capability similar to that of DRAM region but without DMA. This memory will be consumed first per heap initialization order by early startup services and scheduler related code. Speed wise RTC fast memory operates on APB clock and hence does not have much performance impact.

**Memory protection**
Contains:

* CONFIG_ESP_SYSTEM_PMP_IDRAM_SPLIT
* CONFIG_ESP_SYSTEM_MEMPROT_FEATURE

CONFIG_ESP_SYSTEM_PMP_IDRAM_SPLIT
Enable IRAM/DRAM split protection

*Found in: Component config > ESP System Settings > Memory protection*

If enabled, the CPU watches all the memory access and raises an exception in case of any memory violation. This feature automatically splits the SRAM memory, using PMP, into data and instruction segments and sets Read/Execute permissions for the instruction part (below given splitting address) and Read/Write permissions for the data part (above the splitting address). The memory protection is effective on all access through the IRAM0 and DRAM0 buses.

**Default value:**

• Yes (enabled) if SOC_CPU_IDRAM_SPLIT_USING_PMP

CONFIG_ESP_SYSTEM_MEMPROT_FEATURE
Enable memory protection

*Found in: Component config > ESP System Settings > Memory protection*

If enabled, the permission control module watches all the memory access and fires the panic handler if a permission violation is detected. This feature automatically splits the SRAM memory into data
and instruction segments and sets Read/Execute permissions for the instruction part (below given splitting address) and Read/Write permissions for the data part (above the splitting address). The memory protection is effective on all access through the IRAM0 and DRAM0 buses.

Default value:
- Yes (enabled) if SOC_MEMPROT_SUPPORTED

**CONFIG_ESP_SYSTEM_MEMPROT_FEATURE_LOCK**

Lock memory protection settings

*Found in: Component config > ESP System Settings > Memory protection > CONFIG_ESP_SYSTEM_MEMPROT_FEATURE*

Once locked, memory protection settings cannot be changed anymore. The lock is reset only on the chip startup.

Default value:
- Yes (enabled) if CONFIG_ESP_SYSTEM_MEMPROT_FEATURE

**CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE**

System event queue size

*Found in: Component config > ESP System Settings*

Configure system event queue size in different application.

Default value:
- 32

**CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE**

Event loop task stack size

*Found in: Component config > ESP System Settings*

Configure system event task stack size in different application.

Default value:
- 2304

**CONFIG_ESP_MAIN_TASK_STACK_SIZE**

Main task stack size

*Found in: Component config > ESP System Settings*

Configure the “main task” stack size. This is the stack of the task which calls app_main(). If app_main() returns then this task is deleted and its stack memory is freed.

Default value:
- 3584

**CONFIG_ESP_MAIN_TASK_AFFINITY**

Main task core affinity

*Found in: Component config > ESP System Settings*

Configure the “main task” core affinity. This is the used core of the task which calls app_main(). If app_main() returns then this task is deleted.

Available options:
- CPU0 (ESP_MAIN_TASK_AFFINITY_CPU0)
Chapter 2. API

- CPU1 (ESP_MAIN_TASK_AFFINITY_CPU1)
- No affinity (ESP_MAIN_TASK_AFFINITY_NO_AFFINITY)

**CONFIG_ESP_MINIMAL_SHARED_STACK_SIZE**

Minimal allowed size for shared stack

*Found in: Component config > ESP System Settings*

Minimal value of size, in bytes, accepted to execute an expression with shared stack.

*Default value:*
- 2048

**CONFIG_ESP_CONSOLE_UART**

Channel for console output

*Found in: Component config > ESP System Settings*

Select where to send console output (through stdout and stderr).

- Default is to use UART0 on pre-defined GPIOs.
- If “Custom” is selected, UART0 or UART1 can be chosen, and any pins can be selected.
- If “None” is selected, there will be no console output on any UART, except for initial output from ROM bootloader. This ROM output can be suppressed by GPIO strapping or EFUSE, refer to chip datasheet for details.
- On chips with USB OTG peripheral, “USB CDC” option redirects output to the CDC port. This option uses the CDC driver in the chip ROM. This option is incompatible with TinyUSB stack.
- On chips with an USB serial/JTAG debug controller, selecting the option for that redirects output to the CDC/ACM (serial port emulation) component of that device.

*Available options:*
- Default: UART0 (ESP_CONSOLE_UART_DEFAULT)
- USB CDC (ESP_CONSOLE_USB_CDC)
- USB Serial/JTAG Controller (ESP_CONSOLE_USB_SERIAL_JTAG)
- Custom UART (ESP_CONSOLE_UART_CUSTOM)
- None (ESP_CONSOLE_NONE)

**CONFIG_ESP_CONSOLE_SECONDARY**

Channel for console secondary output

*Found in: Component config > ESP System Settings*

This secondary option supports output through other specific port like USB_SERIAL_JTAG when UART0 port as a primary is selected but not connected. This secondary output currently only supports non-blocking mode without using REPL. If you want to output in blocking mode with REPL or input through this secondary port, please change the primary config to this port in Channel for console output menu.

*Available options:*
- No secondary console (ESP_CONSOLE_SECONDARY_NONE)
- USB_SERIAL_JTAG PORT (ESP_CONSOLE_SECONDARY_USB_SERIAL_JTAG)
    This option supports output through USB_SERIAL_JTAG port when the UART0 port is not connected. The output currently only supports non-blocking mode without using the console. If you want to output in blocking mode with REPL or input through USB_SERIAL_JTAG port, please change the primary config to ESP_CONSOLE_USB_SERIAL_JTAG above.
**CONFIG_ESP_CONSOLE_UART_NUM**

UART peripheral to use for console output (0-1)

*Found in: Component config > ESP System Settings*

This UART peripheral is used for console output from the ESP-IDF Bootloader and the app. If the configuration is different in the Bootloader binary compared to the app binary, UART is reconfigured after the bootloader exits and the app starts.

Due to an ESP32 ROM bug, UART2 is not supported for console output via esp_rom_printf.

**Available options:**
- UART0 (ESP_CONSOLE_UART_CUSTOM_NUM_0)
- UART1 (ESP_CONSOLE_UART_CUSTOM_NUM_1)

**CONFIG_ESP_CONSOLE_UART_TX_GPIO**

UART TX on GPIO#

*Found in: Component config > ESP System Settings*

This GPIO is used for console UART TX output in the ESP-IDF Bootloader and the app (including boot log output and default standard output and standard error of the app). If the configuration is different in the Bootloader binary compared to the app binary, UART is reconfigured after the bootloader exits and the app starts.

**Range:**
- from 0 to 46 if ESP_CONSOLE_UART_CUSTOM

**Default value:**
- 1 if ESP_CONSOLE_UART_CUSTOM
- 43 if ESP_CONSOLE_UART_CUSTOM

**CONFIG_ESP_CONSOLE_UART_RX_GPIO**

UART RX on GPIO#

*Found in: Component config > ESP System Settings*

This GPIO is used for UART RX input in the ESP-IDF Bootloader and the app (including default default standard input of the app).

**Note:** The default ESP-IDF Bootloader configures this pin but doesn’t read anything from the UART.

If the configuration is different in the Bootloader binary compared to the app binary, UART is reconfigured after the bootloader exits and the app starts.

**Range:**
- from 0 to 46 if ESP_CONSOLE_UART_CUSTOM

**Default value:**
- 3 if ESP_CONSOLE_UART_CUSTOM
- 44 if ESP_CONSOLE_UART_CUSTOM

**CONFIG_ESP_CONSOLE_UART_BAUDRATE**

UART console baud rate

*Found in: Component config > ESP System Settings*

This baud rate is used by both the ESP-IDF Bootloader and the app (including boot log output and default standard input/output/error of the app).

The app’s maximum baud rate depends on the UART clock source. If Power Management is disabled, the UART clock source is the APB clock and all baud rates in the available range will be sufficiently
accurate. If Power Management is enabled, REF_TICK clock source is used so the baud rate is divided
from 1MHz. Baud rates above 1Mbps are not possible and values between 500Kbps and 1Mbps may
not be accurate.

If the configuration is different in the Bootloader binary compared to the app binary, UART is recon-
figured after the bootloader exits and the app starts.

Range:
  - from 1200 to 4000000 if `CONFIG_PM_ENABLE`
  - from 1200 to 1000000 if `CONFIG_PM_ENABLE`

Default value:
  - 115200

**CONFIG_ESP_INT_WDT**

Interrupt watchdog

*Found in: Component config > ESP System Settings*

This watchdog timer can detect if the FreeRTOS tick interrupt has not been called for a certain time,
either because a task turned off interrupts and did not turn them on for a long time, or because an
interrupt handler did not return. It will try to invoke the panic handler first and failing that reset the SoC.

Default value:
  - Yes (enabled)

**CONFIG_ESP_INT_WDT_TIMEOUT_MS**

Interrupt watchdog timeout (ms)

*Found in: Component config > ESP System Settings > CONFIG_ESP_INT_WDT*

The timeout of the watchdog, in miliseconds. Make this higher than the FreeRTOS tick rate.

Range:
  - from 10 to 10000

Default value:
  - 300 if `CONFIG_SPIRAM` && `CONFIG_ESP_INT_WDT`
  - 800 if `CONFIG_SPIRAM` && `CONFIG_ESP_INT_WDT`

**CONFIG_ESP_INT_WDT_CHECK_CPU1**

Also watch CPU1 tick interrupt

*Found in: Component config > ESP System Settings > CONFIG_ESP_INT_WDT*

Also detect if interrupts on CPU 1 are disabled for too long.

Default value:
  - Yes (enabled) if `CONFIG_ESP_INT_WDT` && `CONFIG_FREERTOS_UNICORE`

**CONFIG_ESP_TASK_WDT_EN**

Enable Task Watchdog Timer

*Found in: Component config > ESP System Settings*

The Task Watchdog Timer can be used to make sure individual tasks are still running. Enabling this
option will enable the Task Watchdog Timer. It can be either initialized automatically at startup or
initialized after startup (see Task Watchdog Timer API Reference)

Default value:
  - Yes (enabled)
CONFIG_ESP_TASK_WDT_INIT
Initialize Task Watchdog Timer on startup

*Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN*

Enabling this option will cause the Task Watchdog Timer to be initialized automatically at startup.

**Default value:**
- Yes (enabled)

CONFIG_ESP_TASK_WDT_PANIC
Invoke panic handler on Task Watchdog timeout

*Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT*

If this option is enabled, the Task Watchdog Timer will be configured to trigger the panic handler when it times out. This can also be configured at run time (see Task Watchdog Timer API Reference)

**Default value:**
- No (disabled)

CONFIG_ESP_TASK_WDT_TIMEOUT_S
Task Watchdog timeout period (seconds)

*Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT*

Timeout period configuration for the Task Watchdog Timer in seconds. This is also configurable at run time (see Task Watchdog Timer API Reference)

**Range:**
- from 1 to 60

**Default value:**
- 5

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0
Watch CPU0 Idle Task

*Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT*

If this option is enabled, the Task Watchdog Timer will watch the CPU0 Idle Task. Having the Task Watchdog watch the Idle Task allows for detection of CPU starvation as the Idle Task not being called is usually a symptom of CPU starvation. Starvation of the Idle Task is detrimental as FreeRTOS household tasks depend on the Idle Task getting some runtime every now and then.

**Default value:**
- Yes (enabled)

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1
Watch CPU1 Idle Task

*Found in: Component config > ESP System Settings > CONFIG_ESP_TASK_WDT_EN > CONFIG_ESP_TASK_WDT_INIT*

If this option is enabled, the Task Watchdog Timer will watch the CPU1 Idle Task.

**Default value:**
- Yes (enabled) if CONFIG_ESP_TASK_WDT_INIT && CONFIG_FREERTOS_UNICORE
**CONFIG_ESP_PANIC_HANDLER_IRAM**

Place panic handler code in IRAM

*Found in: Component config > ESP System Settings*

If this option is disabled (default), the panic handler code is placed in flash not IRAM. This means that if ESP-IDF crashes while flash cache is disabled, the panic handler will automatically re-enable flash cache before running GDB Stub or Core Dump. This adds some minor risk, if the flash cache status is also corrupted during the crash.

If this option is enabled, the panic handler code (including required UART functions) is placed in IRAM. This may be necessary to debug some complex issues with crashes while flash cache is disabled (for example, when writing to SPI flash) or when flash cache is corrupted when an exception is triggered.

**Default value:**
- No (disabled)

---

**CONFIG_ESP_DEBUG_STUBS_ENABLE**

OpenOCD debug stubs

*Found in: Component config > ESP System Settings*

Debug stubs are used by OpenOCD to execute pre-compiled onboard code which does some useful debugging stuff, e.g. GCOV data dump.

**Default value:**
- “COMPILER_OPTIMIZATION_LEVEL_DEBUG” if `CONFIG_ESP32_TRAX` && `ESP32S2_TRAX` && `ESP32S3_TRAX`

---

**CONFIG_ESP_DEBUG_OCDAWARE**

Make exception and panic handlers JTAG/OCD aware

*Found in: Component config > ESP System Settings*

The FreeRTOS panic and unhandled exception handlers can detect a JTAG OCD debugger and instead of panicking, have the debugger stop on the offending instruction.

**Default value:**
- Yes (enabled)

---

**CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL**

Interrupt level to use for Interrupt Watchdog and other system checks

*Found in: Component config > ESP System Settings*

Interrupt level to use for Interrupt Watchdog and other system checks.

**Available options:**
- Level 5 interrupt (`ESP_SYSTEM_CHECK_INT_LEVEL_5`) Using level 5 interrupt for Interrupt Watchdog and other system checks.
- Level 4 interrupt (`ESP_SYSTEM_CHECK_INT_LEVEL_4`) Using level 4 interrupt for Interrupt Watchdog and other system checks.

**Brownout Detector** Contains:
- `CONFIG_ESP_BROWNOUT_DET`
CONFIG_ESP_BROWNOUT_DET

Hardware brownout detect & reset

*Found in: Component config > ESP System Settings > Brownout Detector*

The ESP has a built-in brownout detector which can detect if the voltage is lower than a specific value. If this happens, it will reset the chip in order to prevent unintended behaviour.

**Default value:**
- Yes (enabled)

CONFIG_ESP_BROWNOUT_DET_LVL_SEL

Brownout voltage level

*Found in: Component config > ESP System Settings > Brownout Detector > CONFIG_ESP_BROWNOUT_DET*

The brownout detector will reset the chip when the supply voltage is approximately below this level. Note that there may be some variation of brownout voltage level between each ESP chip.

#The voltage levels here are estimates, more work needs to be done to figure out the exact voltages #of the brownout threshold levels.

**Available options:**
- 2.43V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_0)
- 2.48V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_1)
- 2.58V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_2)
- 2.62V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_3)
- 2.67V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_4)
- 2.70V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_5)
- 2.77V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_6)
- 2.80V +/- 0.05 (ESP_BROWNOUT_DET_LVL_SEL_7)

CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE

Permanently disable BASIC ROM Console

*Found in: Component config > ESP System Settings*

If set, the first time the app boots it will disable the BASIC ROM Console permanently (by burning an eFuse).

Otherwise, the BASIC ROM Console starts on reset if no valid bootloader is read from the flash.

(Enabling secure boot also disables the BASIC ROM Console by default.)

**Default value:**
- No (disabled)

**IPC (Inter-Processor Call)**

Contains:
- CONFIG_ESP_IPC_TASK_STACK_SIZE
- CONFIG_ESP_IPC_USES_CALLERS_PRIORITY

CONFIG_ESP_IPC_TASK_STACK_SIZE

Inter-Processor Call (IPC) task stack size

*Found in: Component config > IPC (Inter-Processor Call)*

Configure the IPC tasks stack size. An IPC task runs on each core (in dual core mode), and allows for cross-core function calls. See IPC documentation for more details. The default IPC stack size should be
enough for most common simple use cases. However, users can increase/decrease the stack size to their needs.

Range:
- from 512 to 65536

Default value:
- 1024

**CONFIG_ESP_IPC_USES_CALLERS_PRIORITY**

IPC runs at caller’s priority

*Found in: Component config > IPC (Inter-Processor Call)*

If this option is not enabled then the IPC task will keep behavior same as prior to that of ESP-IDF v4.0, hence IPC task will run at (configMAX_PRIORITIES - 1) priority.

Default value:
- Yes (enabled) if `CONFIG_FREERTOS_UNICORE`

**High resolution timer (esp_timer)**

Contains:
- `CONFIG_ESP_TIMER_PROFILING`
- `CONFIG_ESP_TIMER_TASK_STACK_SIZE`
- `CONFIG_ESP_TIMER_INTERRUPT_LEVEL`
- `CONFIG_ESP_TIMER_SUPPORTS_ISR_DISPATCH_METHOD`

**CONFIG_ESP_TIMER_PROFILING**

Enable esp_timer profiling features

*Found in: Component config > High resolution timer (esp_timer)*

If enabled, esp_timer_dump will dump information such as number of times the timer was started, number of times the timer has triggered, and the total time it took for the callback to run. This option has some effect on timer performance and the amount of memory used for timer storage, and should only be used for debugging/testing purposes.

Default value:
- No (disabled)

**CONFIG_ESP_TIMER_TASK_STACK_SIZE**

High-resolution timer task stack size

*Found in: Component config > High resolution timer (esp_timer)*

Configure the stack size of “timer_task” task. This task is used to dispatch callbacks of timers created using ets_timer and esp_timer APIs. If you are seeing stack overflow errors in timer task, increase this value.

Note that this is not the same as FreeRTOS timer task. To configure FreeRTOS timer task size, see “FreeRTOS timer task stack size” option in “FreeRTOS” menu.

Range:
- from 2048 to 65536

Default value:
- 3584
Chapter 2. API Reference

**CONFIG_ESP TIMER INTERRUPT LEVEL**

Interrupt level

*Found in: Component config > High resolution timer (esp_timer)*

It sets the interrupt level for esp_timer ISR in range 1..3. A higher level (3) helps to decrease the ISR esp_timer latency.

**Range:**
- from 1 to 3
- from 1 to 1

**Default value:**
- 1

**CONFIG_ESP TIMER SUPPORTS ISR DISPATCH METHOD**

Support ISR dispatch method

*Found in: Component config > High resolution timer (esp_timer)*

Allows using ESP_TIMER_ISR dispatch method (ESP_TIMER_TASK dispatch method is also available). - ESP_TIMER_TASK - Timer callbacks are dispatched from a high-priority esp_timer task. - ESP_TIMER_ISR - Timer callbacks are dispatched directly from the timer interrupt handler. The ISR dispatch can be used, in some cases, when a callback is very simple or need a lower-latency.

**Default value:**
- No (disabled)

**Wi-Fi**

Contains:
- `CONFIG_ESP32_WIFI_ENABLE_WPA3_OWE_STA`
- `CONFIG_ESP32_WIFI_ENABLE_WPA3_SAE`
- `CONFIG_ESP32_WIFI_SOFTAP_BEACON_MAX_LEN`
- `CONFIG_ESP32_WIFI_CACHE_TX_BUFFER_NUM`
- `CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM`
- `CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM`
- `CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM`
- `CONFIG_ESP32_WIFI_STATIC_TX_BUFFER_NUM`
- `CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM`
- `CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE`
- `CONFIG_ESP32_WIFI_SW_COEXIST_ENABLE`
- `CONFIG_ESP32_WIFI_TX_BUFFER`
- `CONFIG_ESP32_WIFI_AMPU_RX_ENABLED`
- `CONFIG_ESP32_WIFI_AMPU_TX_ENABLED`
- `CONFIG_ESP32_WIFI_AMSDU_TX_ENABLED`
- `CONFIG_ESP32_WIFI_CSI_ENABLED`
- `CONFIG ESP_WIFI_FTM_ENABLE`
- `CONFIG ESP_WIFI_GCMP_SUPPORT`
- `CONFIG ESP_WIFI_GMAC_SUPPORT`
- `CONFIG ESP32_WIFI_IRAM_OPT`
- `CONFIG ESP32_WIFI_MGMT_SBUF_NUM`
- `CONFIG ESP32_WIFI_SBPUF_NUM`
- `CONFIG ESP32_WIFI_NVS_ENABLED`
- `CONFIG ESP32_WIFI_RX_IRAM_OPT`
- `CONFIG ESP_WIFI_SLP_BEACON_LOST_OPT`
- `CONFIG ESP_WIFI_SLP_IRAM_OPT`
- `CONFIG ESP_WIFI_SOFTAP_SUPPORT`
- `CONFIG ESP32_WIFI_TASK_CORE_ID`
CONFIG_ESP32_WIFI_SW_COEXIST_ENABLE

Software controls WiFi/Bluetooth coexistence

*Found in: Component config > Wi-Fi*

If enabled, WiFi & Bluetooth coexistence is controlled by software rather than hardware. Recommended for heavy traffic scenarios. Both coexistence configuration options are automatically managed, no user intervention is required. If only Bluetooth is used, it is recommended to disable this option to reduce binary file size.

*Default value:*
- Yes (enabled) if `CONFIG_BT_ENABLED`

CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM

Max number of WiFi static RX buffers

*Found in: Component config > Wi-Fi*

Set the number of WiFi static RX buffers. Each buffer takes approximately 1.6KB of RAM. The static rx buffers are allocated when `esp_wifi_init` is called, they are not freed until `esp_wifi_deinit` is called.

WiFi hardware use these buffers to receive all 802.11 frames. A higher number may allow higher throughput but increases memory use. If `ESP32_WIFI_AMPDU_RX_ENABLED` is enabled, this value is recommended to set equal or bigger than `ESP32_WIFI_RX_BA_WIN` in order to achieve better throughput and compatibility with both stations and APs.

*Range:*
- from 2 to 25

*Default value:*
- 10 if `CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP`
- 16 if `CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP`

CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM

Max number of WiFi dynamic RX buffers

*Found in: Component config > Wi-Fi*

Set the number of WiFi dynamic RX buffers, 0 means unlimited RX buffers will be allocated (provided sufficient free RAM). The size of each dynamic RX buffer depends on the size of the received data frame.

For each received data frame, the WiFi driver makes a copy to an RX buffer and then delivers it to the high layer TCP/IP stack. The dynamic RX buffer is freed after the higher layer has successfully received the data frame.

For some applications, WiFi data frames may be received faster than the application can process them. In these cases we may run out of memory if RX buffer number is unlimited (0).

If a dynamic RX buffer limit is set, it should be at least the number of static RX buffers.

*Range:*
- from 0 to 128 if `CONFIG_LWIP_WND_SCALE`
- from 0 to 1024 if `CONFIG_LWIP_WND_SCALE`

*Default value:*
- 32

CONFIG_ESP32_WIFI_TX_BUFFER

Type of WiFi TX buffers

*Found in: Component config > Wi-Fi*
Select type of WiFi TX buffers:

If “Static” is selected, WiFi TX buffers are allocated when WiFi is initialized and released when WiFi is de-initialized. The size of each static TX buffer is fixed to about 1.6KB.

If “Dynamic” is selected, each WiFi TX buffer is allocated as needed when a data frame is delivered to the WiFi driver from the TCP/IP stack. The buffer is freed after the data frame has been sent by the WiFi driver. The size of each dynamic TX buffer depends on the length of each data frame sent by the TCP/IP layer.

If PSRAM is enabled, “Static” should be selected to guarantee enough WiFi TX buffers. If PSRAM is disabled, “Dynamic” should be selected to improve the utilization of RAM.

Available options:
- Static (ESP32_WIFI_STATIC_TX_BUFFER)
- Dynamic (ESP32_WIFI_DYNAMIC_TX_BUFFER)

**CONFIG_ESP32_WIFI_STATIC_TX_BUFFER_NUM**

Max number of WiFi static TX buffers

*Found in: Component config > Wi-Fi*

Set the number of WiFi static TX buffers. Each buffer takes approximately 1.6KB of RAM. The static RX buffers are allocated when esp_wifi_init() is called, they are not released until esp_wifi_deinit() is called.

For each transmitted data frame from the higher layer TCP/IP stack, the WiFi driver makes a copy of it in a TX buffer. For some applications especially UDP applications, the upper layer can deliver frames faster than WiFi layer can transmit. In these cases, we may run out of TX buffers.

*Range:*
- from 1 to 64 if ESP32_WIFI_STATIC_TX_BUFFER

*Default value:*
- 16 if ESP32_WIFI_STATIC_TX_BUFFER

**CONFIG_ESP32_WIFI_CACHE_TX_BUFFER_NUM**

Max number of WiFi cache TX buffers

*Found in: Component config > Wi-Fi*

Set the number of WiFi cache TX buffer number.

For each TX packet from uplayer, such as LWIP etc, WiFi driver needs to allocate a static TX buffer and makes a copy of uplayer packet. If WiFi driver fails to allocate the static TX buffer, it caches the uplayer packets to a dedicated buffer queue, this option is used to configure the size of the cached TX queue.

*Range:*
- from 16 to 128 if CONFIG_SPIRAM

*Default value:*
- 32 if CONFIG_SPIRAM

**CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM**

Max number of WiFi dynamic TX buffers

*Found in: Component config > Wi-Fi*

Set the number of WiFi dynamic TX buffers. The size of each dynamic TX buffer is not fixed, it depends on the size of each transmitted data frame.
For each transmitted frame from the higher layer TCP/IP stack, the WiFi driver makes a copy of it in a TX buffer. For some applications, especially UDP applications, the upper layer can deliver frames faster than WiFi layer can transmit. In these cases, we may run out of TX buffers.

**Range:**
- from 1 to 128

**Default value:**
- 32

**CONFIG_ESP32_WIFI_CSI_ENABLED**

WiFi CSI (Channel State Information)

*Found in: Component config > Wi-Fi*

Select this option to enable CSI (Channel State Information) feature. CSI takes about CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM KB of RAM. If CSI is not used, it is better to disable this feature in order to save memory.

**Default value:**
- No (disabled)

**CONFIG_ESP32_WIFI_AMPDU_TX_ENABLED**

WiFi AMPDU TX

*Found in: Component config > Wi-Fi*

Select this option to enable AMPDU TX feature

**Default value:**
- Yes (enabled)

**CONFIG_ESP32_WIFI_TX_BA_WIN**

WiFi AMPDU TX BA window size

*Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_AMPDU_TX_ENABLED*

Set the size of WiFi Block Ack TX window. Generally a bigger value means higher throughput but more memory. Most of time we should NOT change the default value unless special reason, e.g. test the maximum UDP TX throughput with iperf etc. For iperf test in shieldbox, the recommended value is 9~12.

**Range:**
- from 2 to 32

**Default value:**
- 6

**CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED**

WiFi AMPDU RX

*Found in: Component config > Wi-Fi*

Select this option to enable AMPDU RX feature

**Default value:**
- Yes (enabled)
CONFIG_ESP32_WIFI_RX_BA_WIN

WiFi AMPDU RX BA window size

*Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED*

Set the size of WiFi Block Ack RX window. Generally a bigger value means higher throughput and better compatibility but more memory. Most of time we should NOT change the default value unless special reason, e.g. test the maximum UDP RX throughput with iperf etc. For iperf test in shieldbox, the recommended value is 9~12. If PSRAM is used and WiFi memory is prefered to allocat in PSRAM first, the default and minimum value should be 16 to achieve better throughput and compatibility with both stations and APs.

**Range:**
- from 2 to 32

**Default value:**
- 6 if `CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP` && `CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED`
- 16 if `CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP` && `CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED`

CONFIG_ESP32_WIFI_AMSDU_TX_ENABLED

WiFi AMSDU TX

*Found in: Component config > Wi-Fi*

Select this option to enable AMSDU TX feature

**Default value:**
- No (disabled) if `CONFIG_SPIRAM`

CONFIG_ESP32_WIFI_NVS_ENABLED

WiFi NVS flash

*Found in: Component config > Wi-Fi*

Select this option to enable WiFi NVS flash

**Default value:**
- Yes (enabled)

CONFIG_ESP32_WIFI_TASK_CORE_ID

WiFi Task Core ID

*Found in: Component config > Wi-Fi*

Pinned WiFi task to core 0 or core 1.

**Available options:**
- Core 0 (ESP32_WIFI_TASK_PINNED_TO_CORE_0)
- Core 1 (ESP32_WIFI_TASK_PINNED_TO_CORE_1)

CONFIG_ESP32_WIFI_SOFTAP_BEACON_MAX_LEN

Max length of WiFi SoftAP Beacon

*Found in: Component config > Wi-Fi*

ESP-MESH utilizes beacon frames to detect and resolve root node conflicts (see documentation). However the default length of a beacon frame can simultaneously hold only five root node identifier structures, meaning that a root node conflict of up to five nodes can be detected at one time. In the occurrence of more root nodes conflict involving more than five root nodes, the conflict resolution process will detect
five of the root nodes, resolve the conflict, and re-detect more root nodes. This process will repeat until all root node conflicts are resolved. However this process can generally take a very long time.

To counter this situation, the beacon frame length can be increased such that more root nodes can be detected simultaneously. Each additional root node will require 36 bytes and should be added on top of the default beacon frame length of 752 bytes. For example, if you want to detect 10 root nodes simultaneously, you need to set the beacon frame length as 932 (752+36*5).

Setting a longer beacon length also assists with debugging as the conflicting root nodes can be identified more quickly.

**Range:**
- from 752 to 1256

**Default value:**
- 752

**CONFIG_ESP32_WIFI_MGMT_SBUF_NUM**

WiFi mgmt short buffer number

*Found in: Component config > Wi-Fi*

Set the number of WiFi management short buffer.

**Range:**
- from 6 to 32

**Default value:**
- 32

**CONFIG_ESP32_WIFI_IRAM_OPT**

WiFi IRAM speed optimization

*Found in: Component config > Wi-Fi*

Select this option to place frequently called Wi-Fi library functions in IRAM. When this option is disabled, more than 10Kbytes of IRAM memory will be saved but Wi-Fi throughput will be reduced.

**Default value:**
- No (disabled) if `CONFIG_BT_ENABLED` && `CONFIG_SPIRAM`
- Yes (enabled)

**CONFIG_ESP32_WIFI_RX_IRAM_OPT**

WiFi RX IRAM speed optimization

*Found in: Component config > Wi-Fi*

Select this option to place frequently called Wi-Fi library RX functions in IRAM. When this option is disabled, more than 17Kbytes of IRAM memory will be saved but Wi-Fi performance will be reduced.

**Default value:**
- No (disabled) if `CONFIG_BT_ENABLED` && `CONFIG_SPIRAM`
- Yes (enabled)

**CONFIG_ESP32_WIFI_ENABLE_WPA3_SAE**

Enable WPA3-Personal

*Found in: Component config > Wi-Fi*

Select this option to allow the device to establish a WPA3-Personal connection with eligible AP’s.

PMF (Protected Management Frames) is a prerequisite feature for a WPA3 connection, it needs to be
explicitly configured before attempting connection. Please refer to the Wi-Fi Driver API Guide for details.

**Default value:**
- Yes (enabled)

**CONFIG_ESP32_WIFI_ENABLE_WPA3_OWE_STA**

Enable OWE STA

*Found in: Component config > Wi-Fi*

Select this option to allow the device to establish OWE connection with eligible AP’s. PMF (Protected Management Frames) is a prerequisite feature for a WPA3 connection, it needs to be explicitly configured before attempting connection. Please refer to the Wi-Fi Driver API Guide for details.

**Default value:**
- Yes (enabled)

**CONFIG_ESP_WIFI_SLP_IRAM_OPT**

WiFi SLP IRAM speed optimization

*Found in: Component config > Wi-Fi*

Select this option to place called Wi-Fi library TBTT process and receive beacon functions in IRAM. Some functions can be put in IRAM either by ESP32_WIFI_IRAM_OPT and ESP32_WIFI_RX_IRAM_OPT, or this one. If already enabled ESP32_WIFI_IRAM_OPT, the other 7.3KB IRAM memory would be taken by this option. If already enabled ESP32_WIFI_RX_IRAM_OPT, the other 1.3KB IRAM memory would be taken by this option. If neither of them are enabled, the other 7.4KB IRAM memory would be taken by this option. Wi-Fi power-save mode average current would be reduced if this option is enabled.

**CONFIG_ESP_WIFI_SLP_DEFAULT_MIN_ACTIVE_TIME**

Minimum active time

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_IRAM_OPT*

The minimum timeout for waiting to receive data, unit: milliseconds.

**Range:**
- from 8 to 60 if `CONFIG_ESP_WIFI_SLP_IRAM_OPT`

**Default value:**
- 50 if `CONFIG_ESP_WIFI_SLP_IRAM_OPT`

**CONFIG_ESP_WIFI_SLP_DEFAULT_MAX_ACTIVE_TIME**

Maximum keep alive time

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_IRAM_OPT*

The maximum time that wifi keep alive, unit: seconds.

**Range:**
- from 10 to 60 if `CONFIG_ESP_WIFI_SLP_IRAM_OPT`

**Default value:**
- 10 if `CONFIG_ESP_WIFI_SLP_IRAM_OPT`
Chapter 2. API

**CONFIG_ESP_WIFI_FTM_ENABLE**

WiFi FTM

*Found in: Component config > Wi-Fi*

Enable feature Fine Timing Measurement for calculating WiFi Round-Trip-Time (RTT).

**CONFIG_ESP_WIFI_FTM_INITIATOR_SUPPORT**

FTM Initiator support

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_FTM_ENABLE*

**Default value:**

- Yes (enabled) if CONFIG_ESP_WIFI_FTM_ENABLE

**CONFIG_ESP_WIFI_FTM RESPONDER_SUPPORT**

FTM Responder support

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_FTM_ENABLE*

**Default value:**

- Yes (enabled) if CONFIG_ESP_WIFI_FTM_ENABLE

**CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE**

Power Management for station at disconnected

*Found in: Component config > Wi-Fi*

Select this option to enable power management for station when disconnected. Chip will do modem-sleep when rf module is not in use any more.

**Default value:**

- Yes (enabled)

**CONFIG_ESP_WIFI_GCMP_SUPPORT**

WiFi GCMP Support(GCMP128 and GCMP256)

*Found in: Component config > Wi-Fi*

Select this option to enable GCMP support. GCMP support is compulsory for WiFi Suite-B support.

**CONFIG_ESP_WIFI_GMAC_SUPPORT**

WiFi GMAC Support(GMAC128 and GMAC256)

*Found in: Component config > Wi-Fi*

Select this option to enable GMAC support. GMAC support is compulsory for WiFi 192 bit certification.

**Default value:**

- No (disabled)

**CONFIG_ESP_WIFI_SOFTAP_SUPPORT**

WiFi SoftAP Support

*Found in: Component config > Wi-Fi*

WiFi module can be compiled without SoftAP to save code size.

**Default value:**
• Yes (enabled)

**CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT**

Wifi sleep optimize when beacon lost

*Found in: Component config > Wi-Fi*

Enable wifi sleep optimization when beacon loss occurs and immediately enter sleep mode when the WiFi module detects beacon loss.

**CONFIG_ESP_WIFI_SLP_BEACON_LOST_TIMEOUT**

Beacon loss timeout

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

Timeout time for close rf phy when beacon loss occurs, Unit: 1024 microsecond.

*Range:*
  • from 5 to 100 if `CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT`

*Default value:*
  • 10 if `CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT`

**CONFIG_ESP_WIFI_SLP_BEACON_LOST_THRESHOLD**

Maximum number of consecutive lost beacons allowed

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

Maximum number of consecutive lost beacons allowed, WiFi keeps Rx state when the number of consecutive beacons lost is greater than the given threshold.

*Range:*
  • from 0 to 8 if `CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT`

*Default value:*
  • 3 if `CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT`

**CONFIG_ESP_WIFI_SLP_PHY_ON_DELTA_EARLY_TIME**

Delta early time for RF PHY on

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

Delta early time for rf phy on, When the beacon is lost, the next rf phy on will be earlier the time specified by the configuration item, Unit: 32 microsecond.

*Range:*
  • from 0 to 100 if `CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT`

*Default value:*
  • 2 if `CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT`

**CONFIG_ESP_WIFI_SLP_PHY_OFF_DELTA_TIMEOUT_TIME**

Delta timeout time for RF PHY off

*Found in: Component config > Wi-Fi > CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT*

Delta timeout time for rf phy off, When the beacon is lost, the next rf phy off will be delayed for the time specified by the configuration item. Unit: 1024 microsecond.

*Range:*
  • from 0 to 8 if `CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT`

*Default value:*

---

Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151
• 2 if CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT

CONFIG_ESP_WIFI_ESPNOW_MAX_ENCRYPT_NUM

Maximum espnow encrypt peers number

Found in: Component config > Wi-Fi

Maximum number of encrypted peers supported by espnow. The number of hardware keys for encryption is fixed. And the espnow and SoftAP share the same hardware keys. So this configuration will affect the maximum connection number of SoftAP. Maximum espnow encrypted peers number + maximum number of connections of SoftAP = Max hardware keys number.

When using ESP mesh, this value should be set to a maximum of 6.

Range:
• from 0 to 16

Default value:
• 6

Core dump Contains:

• CONFIG_ESP_COREDUMP_CHECK_BOOT
• CONFIG_ESP_COREDUMP_DATA_FORMAT
• CONFIG_ESP_COREDUMP_CHECKSUM
• CONFIG_ESP_COREDUMP_TO_FLASH_OR_UART
• CONFIG_ESP_COREDUMP_UART_DELAY
• CONFIG_ESP_COREDUMP_DECODE
• CONFIG_ESP_COREDUMP_MAX_TASKS_NUM
• CONFIG_ESP_COREDUMP_STACK_SIZE

CONFIG_ESP_COREDUMP_TO_FLASH_OR_UART

Data destination

Found in: Component config > Core dump

Select place to store core dump: flash, uart or none (to disable core dumps generation).

Core dumps to Flash are not available if PSRAM is used for task stacks.

If core dump is configured to be stored in flash and custom partition table is used add corresponding entry to your CSV. For examples, please see predefined partition table CSV descriptions in the components/partition_table directory.

Available options:
• Flash (ESP_COREDUMP_ENABLE_TO_FLASH)
• UART (ESP_COREDUMP_ENABLE_TO_UART)
• None (ESP_COREDUMP_ENABLE_TO_NONE)

CONFIG_ESP_COREDUMP_DATA_FORMAT

Core dump data format

Found in: Component config > Core dump

Select the data format for core dump.

Available options:
• Binary format (ESP_COREDUMP_DATA_FORMAT_BIN)
• ELF format (ESP_COREDUMP_DATA_FORMAT_ELF)
**CONFIG_ESP_COREDUMP_CHECKSUM**

Core dump data integrity check

*Found in: Component config > Core dump*

Select the integrity check for the core dump.

**Available options:**
- Use CRC32 for integrity verification (ESP_COREDUMP_CHECKSUM_CRC32)
- Use SHA256 for integrity verification (ESP_COREDUMP_CHECKSUM_SHA256)

**CONFIG_ESP_COREDUMP_CHECK_BOOT**

Check core dump data integrity on boot

*Found in: Component config > Core dump*

When enabled, if any data are found on the flash core dump partition, they will be checked by calculating their checksum.

**Default value:**
- Yes (enabled) if ESP_COREDUMP_ENABLE_TO_FLASH

**CONFIG_ESP_COREDUMP_MAX_TASKS_NUM**

Maximum number of tasks

*Found in: Component config > Core dump*

Maximum number of tasks snapshots in core dump.

**CONFIG_ESP_COREDUMP_UART_DELAY**

Delay before print to UART

*Found in: Component config > Core dump*

Config delay (in ms) before printing core dump to UART. Delay can be interrupted by pressing Enter key.

**Default value:**
- 0 if ESP_COREDUMP_ENABLE_TO_UART

**CONFIG_ESP_COREDUMP_STACK_SIZE**

Reserved stack size

*Found in: Component config > Core dump*

Size of the memory to be reserved for core dump stack. If 0 core dump process will run on the stack of crashed task/ISR, otherwise special stack will be allocated. To ensure that core dump itself will not overflow task/ISR stack set this to the value above 800. NOTE: It eats DRAM.

**CONFIG_ESP_COREDUMP_DECODE**

Handling of UART core dumps in IDF Monitor

*Found in: Component config > Core dump*

**Available options:**
- Decode and show summary (info_corefile) (ESP_COREDUMP_DECODE_INFO)
- Don’t decode (ESP_COREDUMP_DECODE_DISABLE)
Chapter 2. API Reference

FAT Filesystem support
Contains:

- `CONFIG_FATFS_API_ENCODING`
- `CONFIG_FATFS_USE_FASTSEEK`
- `CONFIG_FATFS_LONG_FILENAMES`
- `CONFIG_FATFS_MAX_LFN`
- `CONFIG_FATFS_FS_LOCK`
- `CONFIG_FATFS_VOLUME_COUNT`
- `CONFIG_FATFS_CHOOSE_CODEPAGE`
- `CONFIG_FATFS_ALLOC_PREFER_EXTRAM`
- `CONFIG_FATFS_SECTOR_SIZE`
- `CONFIG_FATFS_TIMEOUT_MS`
- `CONFIG_FATFS_PER_FILE_CACHE`

`CONFIG_FATFS_VOLUME_COUNT`
Number of volumes

*Found in: Component config > FAT Filesystem support*

Number of volumes (logical drives) to use.

**Range:**
- from 1 to 10

**Default value:**
- 2

`CONFIG_FATFS_LONG_FILENAMES`
Long filename support

*Found in: Component config > FAT Filesystem support*

Support long filenames in FAT. Long filename data increases memory usage. FATFS can be configured to store the buffer for long filename data in stack or heap.

**Available options:**
- No long filenames (FATFS_LFN_NONE)
- Long filename buffer in heap (FATFS_LFN_HEAP)
- Long filename buffer on stack (FATFS_LFN_STACK)

`CONFIG_FATFS_SECTOR_SIZE`
Sector size

*Found in: Component config > FAT Filesystem support*

Specify the size of the sector in bytes for FATFS partition generator.

**Available options:**
- 512 (FATFS_SECTOR_512)
- 4096 (FATFS_SECTOR_4096)

`CONFIG_FATFS_CHOOSE_CODEPAGE`
OEM Code Page

*Found in: Component config > FAT Filesystem support*

OEM code page used for file name encodings.

If “Dynamic” is selected, code page can be chosen at runtime using `f_setcp` function. Note that choosing this option will increase application size by ~480kB.
Available options:

- Dynamic (all code pages supported) (FATFS_CODEPAGE_DYNAMIC)
- US (CP437) (FATFS_CODEPAGE_437)
- Arabic (CP720) (FATFS_CODEPAGE_720)
- Greek (CP737) (FATFS_CODEPAGE_737)
- KBL (CP771) (FATFS_CODEPAGE_771)
- Baltic (CP775) (FATFS_CODEPAGE_775)
- Latin 1 (CP850) (FATFS_CODEPAGE_850)
- Latin 2 (CP852) (FATFS_CODEPAGE_852)
- Cyrillic (CP855) (FATFS_CODEPAGE_855)
- Turkish (CP857) (FATFS_CODEPAGE_857)
- Portuguese (CP860) (FATFS_CODEPAGE_860)
- Icelandic (CP861) (FATFS_CODEPAGE_861)
- Hebrew (CP862) (FATFS_CODEPAGE_862)
- Canadian French (CP863) (FATFS_CODEPAGE_863)
- Arabic (CP864) (FATFS_CODEPAGE_864)
- Nordic (CP865) (FATFS_CODEPAGE_865)
- Russian (CP866) (FATFS_CODEPAGE_866)
- Greek 2 (CP869) (FATFS_CODEPAGE_869)
- Japanese (DBCS) (CP932) (FATFS_CODEPAGE_932)
- Simplified Chinese (DBCS) (CP936) (FATFS_CODEPAGE_936)
- Korean (DBCS) (CP949) (FATFS_CODEPAGE_949)
- Traditional Chinese (DBCS) (CP950) (FATFS_CODEPAGE_950)

CONFIG_FATFS_MAX_LFN

Max long filename length

*Found in:* Component config > FAT Filesystem support

Maximum long filename length. Can be reduced to save RAM.

**Range:**
- from 12 to 255

**Default value:**
- 255

CONFIG_FATFS_API_ENCODING

API character encoding

*Found in:* Component config > FAT Filesystem support

Choose encoding for character and string arguments/returns when using FATFS APIs. The encoding of arguments will usually depend on text editor settings.

**Available options:**
- API uses ANSI/OEM encoding (FATFS_API_ENCODING_ANSI_OEM)
- API uses UTF-8 encoding (FATFS_API_ENCODING_UTF_8)

CONFIG_FATFS_FS_LOCK

Number of simultaneously open files protected by lock function

*Found in:* Component config > FAT Filesystem support

This option sets the FATFS configuration value _FS_LOCK. The option _FS_LOCK switches file lock function to control duplicated file open and illegal operation to open objects.

* 0: Disable file lock function. To avoid volume corruption, application should avoid illegal open, remove and rename to the open objects.
* >0: Enable file lock function. The value defines how many files/sub-directories can be opened simultaneously under file lock control.

Note that the file lock control is independent of re-entrancy.

**Range:**
- from 0 to 65535

**Default value:**
- 0

**CONFIG_FATFS_TIMEOUT_MS**
Timeout for acquiring a file lock, ms

*Found in: Component config > FAT Filesystem support*

This option sets FATFS configuration value _FS_TIMEOUT, scaled to milliseconds. Sets the number of milliseconds FATFS will wait to acquire a mutex when operating on an open file. For example, if one task is performing a lengthy operation, another task will wait for the first task to release the lock, and time out after amount of time set by this option.

**Default value:**
- 10000

**CONFIG_FATFS_PER_FILE_CACHE**
Use separate cache for each file

*Found in: Component config > FAT Filesystem support*

This option affects FATFS configuration value _FS_TINY. If this option is set, _FS_TINY is 0, and each open file has its own cache, size of the cache is equal to the _MAX_SS variable (512 or 4096 bytes). This option uses more RAM if more than 1 file is open, but needs less reads and writes to the storage for some operations.

If this option is not set, _FS_TINY is 1, and single cache is used for all open files, size is also equal to _MAX_SS variable. This reduces the amount of heap used when multiple files are open, but increases the number of read and write operations which FATFS needs to make.

**Default value:**
- Yes (enabled)

**CONFIG_FATFS_ALLOC_PREFER_EXTRAM**
Perfer external RAM when allocating FATFS buffers

*Found in: Component config > FAT Filesystem support*

When the option is enabled, internal buffers used by FATFS will be allocated from external RAM. If the allocation from external RAM fails, the buffer will be allocated from the internal RAM. Disable this option if optimizing for performance. Enable this option if optimizing for internal memory size.

**Default value:**
- Yes (enabled) if SPIRAM_USE_CAPS_ALLOC || SPIRAM_USE_MALLOC

**CONFIG_FATFS_USE_FASTSEEK**
Enable fast seek algorithm when using lseek function through VFS FAT

*Found in: Component config > FAT Filesystem support*

The fast seek feature enables fast backward/long seek operations without FAT access by using an in-memory CLMT (cluster link map table). Please note, fast-seek is only allowed for read-mode files, if a
file is opened in write-mode, the seek mechanism will automatically fallback to the default implementation.

**Default value:**
- No (disabled)

**CONFIG_FATFS_FASTSEEK_BUFFER_SIZE**

Fast seek CLMT buffer size

*Found in:* Component config > FAT Filesystem support > CONFIG_FATFS_USE_FASTSEEK

If fast seek algorithm is enabled, this defines the size of CLMT buffer used by this algorithm in 32-bit word units. This value should be chosen based on prior knowledge of maximum elements of each file entry would store.

**Default value:**
- 64 if CONFIG_FATFS_USE_FASTSEEK

**FreeRTOS** Contains:
- **Kernel**
- **Port**

**Kernel** Contains:
- `CONFIG_FREERTOS_CHECK_STACKOVERFLOW`
- `CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY`
- `CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS`
- `CONFIG_FREERTOS_MAX_TASK_NAME_LEN`
- `CONFIG_FREERTOS_IDLE_TASK_STACKSIZE`
- `CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS`
- `CONFIG_FREERTOS_QUEUE_REGISTRY_SIZE`
- `CONFIG_FREERTOS_TASK_NOTIFICATION_ARRAY_ENTRIES`
- `CONFIG_FREERTOS_HZ`
- `CONFIG_FREERTOS_TIMER_QUEUE_LENGTH`
- `CONFIG_FREERTOS_TIMER_TASK_PRIORITY`
- `CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH`
- `CONFIG_FREERTOS_USE_IDLE_HOOK`
- `CONFIG_FREERTOS_OPTIMIZED_SCHEDULER`
- `CONFIG_FREERTOS_USE_TICK_HOOK`
- `CONFIG_FREERTOS_USE_TICKLESS_IDLE`
- `CONFIG_FREERTOS_USE_TRACE_FACILITY`
- `CONFIG_FREERTOS_UNICORE`
- `CONFIG_FREERTOS_SMP`
- `CONFIG_FREERTOS_USE_MINIMAL_IDLE_HOOK`

**CONFIG_FREERTOS_SMP**

Run the Amazon SMP FreeRTOS kernel instead (FEATURE UNDER DEVELOPMENT)

*Found in:* Component config > FreeRTOS > Kernel

Amazon has released an SMP version of the FreeRTOS Kernel which can be found via the following link: https://github.com/FreeRTOS/FreeRTOS-Kernel/tree/smp

IDF has added an experimental port of this SMP kernel located in components/freertos/FreeRTOS-Kernel-SMP. Enabling this option will cause IDF to use the Amazon SMP kernel. Note that THIS FEATURE IS UNDER ACTIVE DEVELOPMENT, users use this at their own risk.
Leaving this option disabled will mean the IDF FreeRTOS kernel is used instead, which is located in: components/freertos/FreeRTOS-Kernel. Both kernel versions are SMP capable, but differ in their implementation and features.

**Default value:**
- No (disabled)

**CONFIG_FREERTOS_UNICORE**

Run FreeRTOS only on first core

*Found in: Component config > FreeRTOS > Kernel*

This version of FreeRTOS normally takes control of all cores of the CPU. Select this if you only want to start it on the first core. This is needed when e.g. another process needs complete control over the second core.

**CONFIG_FREERTOS_HZ**

configTICK_RATE_HZ

*Found in: Component config > FreeRTOS > Kernel*

Sets the FreeRTOS tick interrupt frequency in Hz (see configTICK_RATE_HZ documentation for more details).

**Range:**
- from 1 to 1000

**Default value:**
- 100

**CONFIG_FREERTOS_OPTIMIZED_SCHEDULER**

configUSE_PORT_OPTIMISED_TASK_SELECTION

*Found in: Component config > FreeRTOS > Kernel*

Enables port specific task selection method. This option can speed up the search of ready tasks when scheduling (see configUSE_PORT_OPTIMISED_TASK_SELECTION documentation for more details).

**Default value:**
- Yes (enabled) if CONFIG_FREERTOS_UNICORE

**CONFIG_FREERTOS_CHECK_STACKOVERFLOW**

configCHECK_FOR_STACK_OVERFLOW

*Found in: Component config > FreeRTOS > Kernel*

Enables FreeRTOS to check for stack overflows (see configCHECK_FOR_STACK_OVERFLOW documentation for more details).

**Note:** If users do not provide their own vApplicationStackOverflowHook() function, a default function will be provided by ESP-IDF.

**Available options:**
- No checking (FREERTOS_CHECK_STACKOVERFLOW_NONE)
- Do not check for stack overflows (configCHECK_FOR_STACK_OVERFLOW = 0)
- Check by stack pointer value (Method 1) (FREERTOS_CHECK_STACKOVERFLOW_PTRVAL)
  Check for stack overflows on each context switch by checking if the stack pointer is in a valid range. Quick but does not detect stack overflows that happened between context switches (configCHECK_FOR_STACK_OVERFLOW = 1)
• Check using canary bytes (Method 2) (FREERTOS_CHECK_STACKOVERFLOW_CANARY)

Places some magic bytes at the end of the stack area and on each context switch, check if these bytes are still intact. More thorough than just checking the pointer, but also slightly slower.
(configCHECK_FOR_STACK_OVERFLOW = 2)

CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS

configNUM_THREAD_LOCAL_STORAGE_POINTERS

*Found in: Component config > FreeRTOS > Kernel*

Set the number of thread local storage pointers in each task (see configNUM_THREAD_LOCAL_STORAGE_POINTERS documentation for more details).

*Note: In ESP-IDF, this value must be at least 1. Index 0 is reserved for use by the pthreads API thread-local-storage. Other indexes can be used for any desired purpose.*

*Range:*
  • from 1 to 256

*Default value:*
  • 1

CONFIG_FREERTOS_IDLE_TASK_STACKSIZE

configMINIMAL_STACK_SIZE (Idle task stack size)

*Found in: Component config > FreeRTOS > Kernel*

Sets the idle task stack size in bytes (see configMINIMAL_STACK_SIZE documentation for more details).

*Note:*
  • ESP-IDF specifies stack sizes in bytes instead of words.
  • The default size is enough for most use cases.
  • The stack size may need to be increased above the default if the app installs idle or thread local storage cleanup hooks that use a lot of stack memory.
  • Conversely, the stack size can be reduced to the minimum if none of the idle features are used.

*Range:*
  • from 768 to 32768

*Default value:*
  • 1536

CONFIG_FREERTOS_USE_IDLE_HOOK

configUSE_IDLE_HOOK

*Found in: Component config > FreeRTOS > Kernel*

Enables the idle task application hook (see configUSE_IDLE_HOOK documentation for more details).

*Note:*
  • The application must provide the hook function `void vApplicationIdleHook( void );`
  • `vApplicationIdleHook()` is called from FreeRTOS idle task(s)
  • The FreeRTOS idle hook is NOT the same as the ESP-IDF Idle Hook, but both can be enabled simultaneously.

*Default value:*
  • No (disabled)
**CONFIG_FREERTOS_USE_MINIMAL_IDLE_HOOK**

Use FreeRTOS minimal idle hook

*Found in: Component config > FreeRTOS > Kernel*

Enables the minimal idle task application hook (see configUSE_IDLE_HOOK documentation for more details).

*Note:*
- The application must provide the hook function `void vApplicationMinimalIdleHook(void);`
- `vApplicationMinimalIdleHook()` is called from FreeRTOS minimal idle task(s)

*Default value:*
- No (disabled) if `CONFIG_FREERTOS_SMP`

**CONFIG_FREERTOS_USE_TICK_HOOK**

`configUSE_TICK_HOOK`

*Found in: Component config > FreeRTOS > Kernel*

Enables the tick hook (see configUSE_TICK_HOOK documentation for more details).

*Note:*
- The application must provide the hook function `void vApplicationTickHook(void);`
- `vApplicationTickHook()` is called from FreeRTOS's tick handling function `xTaskIncrementTick()`
- The FreeRTOS tick hook is NOT the same as the ESP-IDF Tick Interrupt Hook, but both can be enabled simultaneously.

*Default value:*
- No (disabled)

**CONFIG_FREERTOS_MAX_TASK_NAME_LEN**

`configMAX_TASK_NAME_LEN`

*Found in: Component config > FreeRTOS > Kernel*

Sets the maximum number of characters for task names (see configMAX_TASK_NAME_LEN documentation for more details).

*Note: For most uses, the default of 16 characters is sufficient.*

*Range:*
- from 1 to 256

*Default value:*
- 16

**CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY**

`configENABLE_BACKWARD_COMPATIBILITY`

*Found in: Component config > FreeRTOS > Kernel*

Enable backward compatibility with APIs prior to FreeRTOS v8.0.0. (see configENABLE_BACKWARD_COMPATIBILITY documentation for more details).

*Default value:*
- No (disabled)
CONFIG_FREERTOS_TIMER_TASK_PRIORITY

`configTIMER_TASK_PRIORITY`

*Found in: Component config > FreeRTOS > Kernel*

Sets the timer task’s priority (see `configTIMER_TASK_PRIORITY` documentation for more details).

**Range:**
- from 1 to 25

**Default value:**
- 1

CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH

`configTIMER_TASK_STACK_DEPTH`

*Found in: Component config > FreeRTOS > Kernel*

Set the timer task’s stack size (see `configTIMER_TASK_STACK_DEPTH` documentation for more details).

**Range:**
- from 1536 to 32768

**Default value:**
- 2048

CONFIG_FREERTOS_TIMER_QUEUE_LENGTH

`configTIMER_QUEUE_LENGTH`

*Found in: Component config > FreeRTOS > Kernel*

Set the timer task’s command queue length (see `configTIMER_QUEUE_LENGTH` documentation for more details).

**Range:**
- from 5 to 20

**Default value:**
- 10

CONFIG_FREERTOS_QUEUE_REGISTRY_SIZE

`configQUEUE_REGISTRY_SIZE`

*Found in: Component config > FreeRTOS > Kernel*

Set the size of the queue registry (see `configQUEUE_REGISTRY_SIZE` documentation for more details).

Note: A value of 0 will disable queue registry functionality

**Range:**
- from 0 to 20

**Default value:**
- 0

CONFIG_FREERTOS_TASK_NOTIFICATION_ARRAY_ENTRIES

`configTASK_NOTIFICATION_ARRAY_ENTRIES`

*Found in: Component config > FreeRTOS > Kernel*
Set the size of the task notification array of each task. When increasing this value, keep in mind that this means additional memory for each and every task on the system. However, task notifications in general are more lightweight compared to alternatives such as semaphores.

Range:
- from 1 to 32

Default value:
- 1

**CONFIG_FREERTOS_USE_TRACE_FACILITY**

configUSE_TRACE_FACILITY

*Found in: Component config > FreeRTOS > Kernel*

Enables additional structure members and functions to assist with execution visualization and tracing (see configUSE_TRACE_FACILITY documentation for more details).

Default value:
- No (disabled)

**CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS**

cfgUSE_STATS_FORMATTING_FUNCTIONS

*Found in: Component config > FreeRTOS > Kernel > CONFIG_FREERTOS_USE_TRACE_FACILITY*

Set configUSE_TRACE_FACILITY and configUSE_STATS_FORMATTING_FUNCTIONS to 1 to include the vTaskList() and vTaskGetRunTimeStats() functions in the build (see configUSE_STATS_FORMATTING_FUNCTIONS documentation for more details).

Default value:
- No (disabled) if CONFIG_FREERTOS_USE_TRACE_FACILITY

**CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID**

Enable display of xCoreID in vTaskList

*Found in: Component config > FreeRTOS > Kernel > CONFIG_FREERTOS_USE_TRACE_FACILITY > CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS*

If enabled, this will include an extra column when vTaskList is called to display the CoreID the task is pinned to (0,1) or -1 if not pinned.

Default value:
- No (disabled) if CONFIG_FREERTOS_SMP && CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS

**CONFIG_FREERTOSGENERATE_RUN_TIME_STATS**

cfgGENERATE_RUN_TIME_STATS

*Found in: Component config > FreeRTOS > Kernel*

Enables collection of run time statistics for each task (see configGENERATE_RUN_TIME_STATS documentation for more details).

Note: The clock used for run time statistics can be configured in FREERTOS_RUN_TIME_STATS_CLK.

Default value:
- No (disabled)
Chapter 2. API 参考

**CONFIG_FREERTOS_USE_TICKLESS_IDLE**

```c
configUSE_TICKLESS_IDLE
```

*Found in: Component config > FreeRTOS > Kernel*

If power management support is enabled, FreeRTOS will be able to put the system into light sleep mode when no tasks need to run for a number of ticks. This number can be set using `FREERTOS_IDLE_TIME_BEFORE_SLEEP` option. This feature is also known as “automatic light sleep”.

Note that timers created using esp_timer APIs may prevent the system from entering sleep mode, even when no tasks need to run. To skip unnecessary wake-up initialize a timer with the “skip_unhandled_events” option as true.

If disabled, automatic light sleep support will be disabled.

*Default value:*
  - No (disabled) if `CONFIG_PM_ENABLE`

**CONFIG_FREERTOS_IDLE_TIME_BEFORE_SLEEP**

```c
configEXPECTED_IDLE_TIME_BEFORE_SLEEP
```

*Found in: Component config > FreeRTOS > Kernel > CONFIG_FREERTOS_USE_TICKLESS_IDLE*

FreeRTOS will enter light sleep mode if no tasks need to run for this number of ticks.

*Range:*
  - from 2 to 4294967295 if `CONFIG_FREERTOS_USE_TICKLESS_IDLE`

*Default value:*
  - 3 if `CONFIG_FREERTOS_USE_TICKLESS_IDLE`

**Port**

Contains:

- `CONFIG_FREERTOS_CHECK_MUTEX_GIVEN_BY_OWNER`
- `CONFIG_FREERTOS_RUN_TIME_STATS_CLK`
- `CONFIG_FREERTOS_INTERRUPT_BACKTRACE`
- `CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK`
- `CONFIG_FREERTOS_ENABLE_STATIC_TASK_CLEAN_UP`
- `CONFIG_FREERTOS_ENABLE_TASK_SNAPSHOT`
- `CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS`
- `CONFIG_FREERTOS_ISR_STACKSIZE`
- `CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH`
- `CONFIG_FREERTOS_PLACE_SNAPSHOT_FUNS_INTO_FLASH`
- `CONFIG_FREERTOS_CHECK_PORT_CRITICAL_COMPLIANCE`
- `CONFIG_FREERTOS_CORETIMER`
- `CONFIG_FREERTOS_FPU_IN_ISR`
- `CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER`

**CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER**

Wrap task functions

*Found in: Component config > FreeRTOS > Port*

If enabled, all FreeRTOS task functions will be enclosed in a wrapper function. If a task function mistakenly returns (i.e. does not delete), the call flow will return to the wrapper function. The wrapper function will then log an error and abort the application. This option is also required for GDB backtraces and C++ exceptions to work correctly inside top-level task functions.

*Default value:*
  - Yes (enabled)
CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK

Enable stack overflow debug watchpoint

*Found in: Component config > FreeRTOS > Port*

FreeRTOS can check if a stack has overflowed its bounds by checking either the value of the stack pointer or by checking the integrity of canary bytes. (See FREERTOS_CHECK_STACKOVERFLOW for more information.) These checks only happen on a context switch, and the situation that caused the stack overflow may already be long gone by then. This option will use the last debug memory watchpoint to allow breaking into the debugger (or panic) as soon as any of the last 32 bytes on the stack of a task are overwritten. The side effect is that using gdb, you effectively have one hardware watchpoint less because the last one is overwritten as soon as a task switch happens.

Another consequence is that due to alignment requirements of the watchpoint, the usable stack size decreases by up to 60 bytes. This is because the watchpoint region has to be aligned to its size and the size for the stack watchpoint in IDF is 32 bytes.

This check only triggers if the stack overflow writes within 32 bytes near the end of the stack, rather than overshooting further, so it is worth combining this approach with one of the other stack overflow check methods.

When this watchpoint is hit, gdb will stop with a SIGTRAP message. When no JTAG OCD is attached, esp-idf will panic on an unhandled debug exception.

Default value:
- No (disabled)

CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS

Enable thread local storage pointers deletion callbacks

*Found in: Component config > FreeRTOS > Port*

ESP-IDF provides users with the ability to free TLSP memory by registering TLSP deletion callbacks. These callbacks are automatically called by FreeRTOS when a task is deleted. When this option is turned on, the memory reserved for TLSPs in the TCB is doubled to make space for storing the deletion callbacks. If the user does not wish to use TLSP deletion callbacks then this option could be turned off to save space in the TCB memory.

Default value:
- Yes (enabled)

CONFIG_FREERTOS_ENABLE_STATIC_TASK_CLEAN_UP

Enable static task clean up hook

*Found in: Component config > FreeRTOS > Port*

Enable this option to make FreeRTOS call the static task clean up hook when a task is deleted.

Note: Users will need to provide a `void vPortCleanUpTCB(void *pxTCB)` callback

Default value:
- No (disabled)

CONFIG_FREERTOS_CHECK_MUTEX_GIVEN_BY_OWNER

Check that mutex semaphore is given by owner task

*Found in: Component config > FreeRTOS > Port*

If enabled, assert that when a mutex semaphore is given, the task giving the semaphore is the task which is currently holding the mutex.

Default value:
Chapter 2. API

• Yes (enabled) if CONFIG_FREERTOS_SMP

CONFIG_FREERTOS_ISR_STACKSIZE

ISR stack size

Found in: Component config > FreeRTOS > Port

The interrupt handlers have their own stack. The size of the stack can be defined here. Each processor has its own stack, so the total size occupied will be twice this.

Range:
  • from 2096 to 32768 if ESP_COREDUMP_DATA_FORMAT_ELF
  • from 1536 to 32768

Default value:
  • 2096 if ESP_COREDUMP_DATA_FORMAT_ELF
  • 1536

CONFIG_FREERTOS_INTERRUPT_BACKTRACE

Enable backtrace from interrupt to task context

Found in: Component config > FreeRTOS > Port

If this option is enabled, interrupt stack frame will be modified to point to the code of the interrupted task as its return address. This helps the debugger (or the panic handler) show a backtrace from the interrupt to the task which was interrupted. This also works for nested interrupts: higher level interrupt stack can be traced back to the lower level interrupt. This option adds 4 instructions to the interrupt dispatching code.

Default value:
  • Yes (enabled)

CONFIG_FREERTOS_FPU_IN_ISR

Use float in Level 1 ISR

Found in: Component config > FreeRTOS > Port

When enabled, the usage of float type is allowed inside Level 1 ISRs. Note that usage of float types in higher level interrupts is still not permitted.

Default value:
  • No (disabled)

CONFIG_FREERTOS_CORETIMER

Tick timer source (Xtensa Only)

Found in: Component config > FreeRTOS > Port

FreeRTOS needs a timer with an associated interrupt to use as the main tick source to increase counters, run timers and do pre-emptive multitasking with. There are multiple timers available to do this, with different interrupt priorities.

Available options:
  • Timer 0 (int 6, level 1) (FREERTOS_CORETIMER_0)
    Select this to use timer 0
  • Timer 1 (int 15, level 3) (FREERTOS_CORETIMER_1)
    Select this to use timer 1
  • SYSTIMER 0 (level 1) (FREERTOS_CORETIMER_SYSTIMER_LVL1)
    Select this to use systimer with the 1 interrupt priority.
Chapter 2. API

- SYSTIMER 0 (level 3) (FREERTOS_CORETIMER_SYSTIMER_LVL3)
  Select this to use systimer with the 3 interrupt priority.

**CONFIG_FREERTOS_RUN_TIME_STATS_CLK**

Choose the clock source for run time stats

*Found in:* Component config > FreeRTOS > Port

Choose the clock source for FreeRTOS run time stats. Options are CPU0’s CPU Clock or the ESP Timer. Both clock sources are 32 bits. The CPU Clock can run at a higher frequency hence provide a finer resolution but will overflow much quicker. Note that run time stats are only valid until the clock source overflows.

**Available options:**

- Use ESP TIMER for run time stats (FREERTOS_RUN_TIME_STATS_USING_ESP_TIMER) ESP Timer will be used as the clock source for FreeRTOS run time stats. The ESP Timer runs at a frequency of 1MHz regardless of Dynamic Frequency Scaling. Therefore the ESP Timer will overflow in approximately 4290 seconds.

- Use CPU Clock for run time stats (FREERTOS_RUN_TIME_STATS_USING_CPU_CLK) CPU Clock will be used as the clock source for the generation of run time stats. The CPU Clock has a frequency dependent on ESP_DEFAULT_CPU_FREQ_MHZ and Dynamic Frequency Scaling (DFS). Therefore the CPU Clock frequency can fluctuate between 80 to 240MHz. Run time stats generated using the CPU Clock represents the number of CPU cycles each task is allocated and DOES NOT reflect the amount of time each task runs for (as CPU clock frequency can change). If the CPU clock consistently runs at the maximum frequency of 240MHz, it will overflow in approximately 17 seconds.

**CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH**

Place FreeRTOS functions into Flash

*Found in:* Component config > FreeRTOS > Port

When enabled the selected Non-ISR FreeRTOS functions will be placed into Flash memory instead of IRAM. This saves up to 8KB of IRAM depending on which functions are used.

**Default value:**

- No (disabled)

**CONFIG_FREERTOS_PLACE_SNAPSHOT_FUNS_INTO_FLASH**

Place task snapshot functions into flash

*Found in:* Component config > FreeRTOS > Port

When enabled, the functions related to snapshots, such as vTaskGetSnapshot or uxTaskGetSnapshotAll, will be placed in flash. Note that if enabled, these functions cannot be called when cache is disabled.

**Default value:**

- No (disabled) if CONFIG_FREERTOS_ENABLE_TASK_SNAPSHOT && CONFIG_ESP_PANIC_HANDLER_IRAM

**CONFIG_FREERTOS_CHECK_PORT_CRITICAL_COMPLIANCE**

Tests compliance with Vanilla FreeRTOS port*._CRITICAL calls

*Found in:* Component config > FreeRTOS > Port

If enabled, context of port*._CRITICAL calls (ISR or Non-ISR) would be checked to be in compliance with Vanilla FreeRTOS. e.g Calling port*._CRITICAL from ISR context would cause assert failure

**Default value:**
• No (disabled)

**CONFIG_FREERTOS_ENABLE_TASK_SNAPSHOT**

Enable task snapshot functions

*Found in: Component config > FreeRTOS > Port*

When enabled, the functions related to snapshots, such as vTaskGetSnapshot or uxTaskGetSnapshotAll, are compiled and linked. Task snapshots are used by Task Watchdog (TWDT), GDB Stub and Core dump.

**Default value:**

• Yes (enabled)

**Hardware Abstraction Layer (HAL) and Low Level (LL)**

Contains:

• **CONFIG_HAL_DEFAULT_ASSERTION_LEVEL**
  
  Default HAL assertion level

*Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)*

Set the assert behavior / level for HAL component. HAL component assert level can be set separately, but the level can’t exceed the system assertion level. e.g. If the system assertion is disabled, then the HAL assertion can’t be enabled either. If the system assertion is enable, then the HAL assertion can still be disabled by this Kconfig option.

**Available options:**

• Same as system assertion level (HAL_ASSERTION_EQUALS_SYSTEM)
• Disabled (HAL_ASSERTION_DISABLE)
• Silent (HAL_ASSERTION_SILENT)
• Enabled (HAL_ASSERTION_ENABLE)

• **CONFIG_HAL_LOG_LEVEL**

  HAL layer log verbosity

*Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)*

Specify how much output to see in HAL logs.

**Available options:**

• No output (HAL_LOG_LEVEL_NONE)
• Error (HAL_LOG_LEVEL_ERROR)
• Warning (HAL_LOG_LEVEL_WARN)
• Info (HAL_LOG_LEVEL_INFO)
• Debug (HAL_LOG_LEVEL_DEBUG)
• Verbose (HAL_LOG_LEVEL_VERBOSE)

• **CONFIG_HAL_SYSTIMER_USE_ROM IMPL**

  Use ROM implementation of SysTimer HAL driver

*Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)*

Enable this flag to use HAL functions from ROM instead of ESP-IDF.
If keeping this as “n” in your project, you will have less free IRAM. If making this as “y” in your project, you will increase free IRAM, but you will lose the possibility to debug this module, and some new features will be added and bugs will be fixed in the IDF source but cannot be synced to ROM.

**Default value:**
- Yes (enabled) if ESP_ROM_HAS_HAL_SYSTIMER

**CONFIG_HAL_WDT_USE_ROM_IMPL**

Use ROM implementation of WDT HAL driver

*Found in: Component config > Hardware Abstraction Layer (HAL) and Low Level (LL)*

Enable this flag to use HAL functions from ROM instead of ESP-IDF.

If keeping this as “n” in your project, you will have less free IRAM. If making this as “y” in your project, you will increase free IRAM, but you will lose the possibility to debug this module, and some new features will be added and bugs will be fixed in the IDF source but cannot be synced to ROM.

**Default value:**
- Yes (enabled) if ESP_ROM_HAS_HAL_WDT

**Heap memory debugging**  
Contains:

- `CONFIG_HEAP_ABORT_WHEN_ALLOCATION_FAILS`
- `CONFIG_HEAP_TASK_TRACKING`
- `CONFIG_HEAP_CORRUPTION_DETECTION`
- `CONFIG_HEAP_TRACING_DEST`
- `CONFIG_HEAP_TRACING_STACK_DEPTH`
- `CONFIG_HEAP_TLSF_USE_ROM_IMPL`

**CONFIG_HEAP_CORRUPTION_DETECTION**

Heap corruption detection

*Found in: Component config > Heap memory debugging*

Enable heap poisoning features to detect heap corruption caused by out-of-bounds access to heap memory.

See the “Heap Memory Debugging” page of the IDF documentation for a description of each level of heap corruption detection.

**Available options:**
- Basic (no poisoning) (HEAP_POISONING_DISABLED)
- Light impact (HEAP_POISONING_LIGHT)
- Comprehensive (HEAP_POISONING_COMPREHENSIVE)

**CONFIG_HEAP_TRACING_DEST**

Heap tracing

*Found in: Component config > Heap memory debugging*

Enables the heap tracing API defined in esp_heap_trace.h.

This function causes a moderate increase in IRAM code side and a minor increase in heap function (malloc/free/realloc) CPU overhead, even when the tracing feature is not used. So it’s best to keep it disabled unless tracing is being used.

**Available options:**
- Disabled (HEAP_TRACING_OFF)
- Standalone (HEAP_TRACING_STANDALONE)
- Host-based (HEAP_TRACING_TOHOST)
CONFIG_HEAP_TRACING_STACK_DEPTH

Heap tracing stack depth

*Found in: Component config > Heap memory debugging*

Number of stack frames to save when tracing heap operation callers.

More stack frames use more memory in the heap trace buffer (and slows down allocation), but can provide useful information.

CONFIG_HEAP_TASK_TRACKING

Enable heap task tracking

*Found in: Component config > Heap memory debugging*

Enables tracking the task responsible for each heap allocation.

This function depends on heap poisoning being enabled and adds four more bytes of overhead for each block allocated.

CONFIG_HEAP_ABORT_WHEN_ALLOCATION_FAILS

Abort if memory allocation fails

*Found in: Component config > Heap memory debugging*

When enabled, if a memory allocation operation fails it will cause a system abort.

Default value:
- No (disabled)

CONFIG_HEAP_TLSF_USE_ROM_IMPL

Use ROM implementation of heap tlsf library

*Found in: Component config > Heap memory debugging*

Enable this flag to use heap functions from ROM instead of ESP-IDF.

If keeping this as “n” in your project, you will have less free IRAM. If making this as “y” in your project, you will increase free IRAM, but you will lose the possibility to debug this module, and some new features will be added and bugs will be fixed in the IDF source but cannot be synced to ROM.

Default value:
- Yes (enabled) if ESP_ROM_HAS_HEAP_TLSF

Log output Contains:

- CONFIG_LOG_DEFAULT_LEVEL
- CONFIG_LOG_TIMESTAMP_SOURCE
- CONFIG_LOG_MAXIMUM_LEVEL
- CONFIG_LOG_COLORS

CONFIG_LOG_DEFAULT_LEVEL

Default log verbosity

*Found in: Component config > Log output*

Specify how much output to see in logs by default. You can set lower verbosity level at runtime using esp_log_level_set function.
By default, this setting limits which log statements are compiled into the program. For example, selecting “Warning” would mean that changing log level to “Debug” at runtime will not be possible. To allow increasing log level above the default at runtime, see the next option.

**Available options:**
- No output (LOG_DEFAULT_LEVEL_NONE)
- Error (LOG_DEFAULT_LEVEL_ERROR)
- Warning (LOG_DEFAULT_LEVEL_WARN)
- Info (LOG_DEFAULT_LEVEL_INFO)
- Debug (LOG_DEFAULT_LEVEL_DEBUG)
- Verbose (LOG_DEFAULT_LEVEL_VERBOSE)

**CONFIG_LOG_MAXIMUM_LEVEL**

Maximum log verbosity

*Found in: Component config > Log output*

This config option sets the highest log verbosity that it’s possible to select at runtime by calling esp_log_level_set(). This level may be higher than the default verbosity level which is set when the app starts up.

This can be used to enable debugging output only at a critical point, for a particular tag, or to minimize start up time but then enable more logs once the firmware has loaded.

Note that increasing the maximum available log level will increase the firmware binary size.

This option only applies to logging from the app, the bootloader log level is fixed at compile time to the separate “Boostrap log verbosity” setting.

**Available options:**
- Same as default (LOG_MAXIMUM_EQUALS_DEFAULT)
- Error (LOG_MAXIMUM_LEVEL_ERROR)
- Warning (LOG_MAXIMUM_LEVEL_WARN)
- Info (LOG_MAXIMUM_LEVEL_INFO)
- Debug (LOG_MAXIMUM_LEVEL_DEBUG)
- Verbose (LOG_MAXIMUM_LEVEL_VERBOSE)

**CONFIG_LOG_COLORS**

Use ANSI terminal colors in log output

*Found in: Component config > Log output*

Enable ANSI terminal color codes in bootloader output.

In order to view these, your terminal program must support ANSI color codes.

**Default value:**
- Yes (enabled)

**CONFIG_LOG_TIMESTAMP_SOURCE**

Log Timestamps

*Found in: Component config > Log output*

Choose what sort of timestamp is displayed in the log output:
- Milliseconds since boot is calculated from the RTOS tick count multiplied by the tick period. This time will reset after a software reboot. e.g. (90000)
- System time is taken from POSIX time functions which use the chip’s RTC and high resolution timers to maintain an accurate time. The system time is initialized to 0 on startup, it can be set with an SNTTP sync, or with POSIX time functions. This time will not reset after a software reboot. e.g. (00:01:30.000)
• NOTE: Currently this will not get used in logging from binary blobs (i.e WiFi & Bluetooth libraries), these will always print milliseconds since boot.

**Available options:**

• Milliseconds Since Boot (LOG_TIMESTAMP_SOURCE_RTS)
• System Time (LOG_TIMESTAMP_SOURCE_SYSTEM)

**LWIP** Contains:

• Checksums
• DHCP server
  • CONFIG_LWIP_DHCP_OPTIONS_LEN
  • CONFIG_LWIP_DHCP_DISABLE_CLIENT_ID
  • CONFIG_LWIP_DHCP_DISABLE_VENDOR_CLASS_ID
  • CONFIG_LWIP_DHCP_DOES_ARP_CHECK
  • CONFIG_LWIP_DHCP_RESTORE_LAST_IP
  • CONFIG_LWIP_PPP_CHAP_SUPPORT
  • CONFIG_LWIP_L2_TO_L3_COPY
  • CONFIG_LWIP_IPV6_DHCP6
  • CONFIG_LWIP_IP4_FRAG
  • CONFIG_LWIP_IP6_FRAG
  • CONFIG_LWIP_IP_FORWARD
  • CONFIG_LWIP_NETBUF_RECVINFO
  • CONFIG_LWIP_AUTOIP
  • CONFIG_LWIP_IPV6
  • CONFIG_LWIP_ENABLE_LCP_ECHO
  • CONFIG_LWIP_ESP_LWIP_ASSERT
  • CONFIG_LWIP_DEBUG
  • CONFIG_LWIP_IRAM_OPTIMIZATION
  • CONFIG_LWIP_STATS
  • CONFIG_LWIP_TIMERS_ONDEMAND
  • CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES
  • CONFIG_LWIP_PPP_MPPE_SUPPORT
  • CONFIG_LWIP_PPP_MSCHAP_SUPPORT
  • CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT
  • CONFIG_LWIP_PPP_PAP_SUPPORT
  • CONFIG_LWIP_PPP_DEBUG_ON
  • CONFIG_LWIP_PPP_SUPPORT
  • CONFIG_LWIP_IP4_REASSEMBLY
  • CONFIG_LWIP_IP6_REASSEMBLY
  • CONFIG_LWIP_SLIP_SUPPORT
  • CONFIG_LWIP_SO_LINGER
  • CONFIG_LWIP_SO_REUSE
  • CONFIG_LWIP_NETIF_API
  • Hooks
  • ICMP
  • CONFIG_LWIP_LOCAL_HOSTNAME
  • LWIP RAW API
  • CONFIG_LWIP_IPV6_NUM_NEIGHBORS
  • CONFIG_LWIP_IPV6_MEMP_NUM_QUEUE
  • CONFIG_LWIP_MAX_SOCKETS
  • CONFIG_LWIP_BRIDGEIF_MAX_PORTS
  • CONFIG_LWIP_NUM_NETIF_CLIENT_DATA
  • CONFIG_LWIP_ESP_GETUFOURS_ARP
  • SNTP
• CONFIG_LWIP_USE_ONLY_LWIP_SELECT
• CONFIG_LWIP_NETIF_LOOPBACK
• TCP
• CONFIG_LWIP_TCP_IP_TASK_AFFINITY
• CONFIG_LWIP_TCP_IP_TASK_STACK_SIZE
• CONFIG_LWIP_TCP_IP_RECVMBOX_SIZE
• UDP
• CONFIG_LWIP_IPV6_RDNSS_MAX_DNS_SERVERS

CONFIG_LWIP_LOCAL_HOSTNAME

Local netif hostname

*Found in: Component config > LWIP*

The default name this device will report to other devices on the network. Could be updated at runtime with esp_netif_set_hostname()

*Default value:*

- "espressif"

CONFIG_LWIP_NETIF_API

Enable usage of standard POSIX APIs in LWIP

*Found in: Component config > LWIP*

If this feature is enabled, standard POSIX APIs: if_indextoname(), if_nametoindex() could be used to convert network interface index to name instead of IDF specific esp-netif APIs (such as esp_netif_get_netif_impl_name())

*Default value:*

- No (disabled)

CONFIG_LWIP_TCP_IP_CORE_LOCKING

Enable tcpip core locking

*Found in: Component config > LWIP*

If Enable tcpip core locking, creates a global mutex that is held during TCPIP thread operations. Can be locked by client code to perform lwIP operations without changing into TCPIP thread using callbacks. See LOCK_TCPIP_CORE() and UNLOCK_TCPIP_CORE().

If disable tcpip core locking, TCP IP will perform tasks through context switching.

*Default value:*

- No (disabled)

CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES

Enable mDNS queries in resolving host name

*Found in: Component config > LWIP*

If this feature is enabled, standard API such as gethostbyname support .local addresses by sending one shot multicast mDNS query

*Default value:*

- Yes (enabled)
CONFIG_LWIP_L2_TO_L3_COPY

Enable copy between Layer2 and Layer3 packets

*Found in: Component config > LWIP*

If this feature is enabled, all traffic from layer2 (WIFI Driver) will be copied to a new buffer before sending it to layer3 (LWIP stack), freeing the layer2 buffer. Please be notified that the total layer2 receiving buffer is fixed and ESP32 currently supports 25 layer2 receiving buffer, when layer2 buffer runs out of memory, then the incoming packets will be dropped in hardware. The layer3 buffer is allocated from the heap, so the total layer3 receiving buffer depends on the available heap size, when heap runs out of memory, no copy will be sent to layer3 and packet will be dropped in layer2. Please make sure you fully understand the impact of this feature before enabling it.

*Default value:*
- No (disabled)

CONFIG_LWIP_IRAM_OPTIMIZATION

Enable LWIP IRAM optimization

*Found in: Component config > LWIP*

If this feature is enabled, some functions relating to RX/TX in LWIP will be put into IRAM, it can improve UDP/TCP throughput by >10% for single core mode, it doesn’t help too much for dual core mode. On the other hand, it needs about 10KB IRAM for these optimizations.

If this feature is disabled, all lwip functions will be put into FLASH.

*Default value:*
- No (disabled)

CONFIG_LWIP_TIMERS_ONDEMAND

Enable LWIP Timers on demand

*Found in: Component config > LWIP*

If this feature is enabled, IGMP and MLD6 timers will be activated only when joining groups or receiving QUERY packets.

This feature will reduce the power consumption for applications which do not use IGMP and MLD6.

*Default value:*
- Yes (enabled)

CONFIG_LWIP_MAX_SOCKETS

Max number of open sockets

*Found in: Component config > LWIP*

Sockets take up a certain amount of memory, and allowing fewer sockets to be open at the same time conserves memory. Specify the maximum amount of sockets here. The valid value is from 1 to 16.

*Range:*
- from 1 to 16

*Default value:*
- 10
CONFIG_LWIP_USE_ONLY_LWIP_SELECT

Support LWIP socket select() only (DEPRECATED)

*Found in: Component config > LWIP*

This option is deprecated. Do not use this option, use VFS_SUPPORT_SELECT instead.

**Default value:**

- No (disabled)

CONFIG_LWIP_SO_LINGER

Enable SO_LINGER processing

*Found in: Component config > LWIP*

Enabling this option allows SO_LINGER processing. l_onoff = 1, l_linger can set the timeout.

If l_linger=0, When a connection is closed, TCP will terminate the connection. This means that TCP will discard any data packets stored in the socket send buffer and send an RST to the peer.

If l_linger!=0, Then closesocket() calls to block the process until the remaining data packets has been sent or timed out.

**Default value:**

- No (disabled)

CONFIG_LWIP_SO_REUSE

Enable SO_REUSEADDR option

*Found in: Component config > LWIP*

Enabling this option allows binding to a port which remains in TIME_WAIT.

**Default value:**

- Yes (enabled)

CONFIG_LWIP_SO_REUSE_RXTOALL

SO_REUSEADDR copies broadcast/multicast to all matches

*Found in: Component config > LWIP > CONFIG_LWIP_SO_REUSE*

Enabling this option means that any incoming broadcast or multicast packet will be copied to all of the local sockets that it matches (may be more than one if SO_REUSEADDR is set on the socket.)

This increases memory overhead as the packets need to be copied, however they are only copied per matching socket. You can safely disable it if you don’t plan to receive broadcast or multicast traffic on more than one socket at a time.

**Default value:**

- Yes (enabled)

CONFIG_LWIP_SO_RCVBUF

Enable SO_RCVBUF option

*Found in: Component config > LWIP*

Enabling this option allows checking for available data on a netconn.

**Default value:**

- No (disabled)
Chapter 2. API 参考

CONFIG_LWIP_NETBUF_RECVINFO

Enable IP_PKTINFO option

*Found in: Component config > LWIP*

Enabling this option allows checking for the destination address of a received IPv4 Packet.

**Default value:**
- No (disabled)

CONFIG_LWIP_IP4_FRAG

Enable fragment outgoing IP4 packets

*Found in: Component config > LWIP*

Enabling this option allows fragmenting outgoing IP4 packets if their size exceeds MTU.

**Default value:**
- Yes (enabled)

CONFIG_LWIP_IP6_FRAG

Enable fragment outgoing IP6 packets

*Found in: Component config > LWIP*

Enabling this option allows fragmenting outgoing IP6 packets if their size exceeds MTU.

**Default value:**
- Yes (enabled)

CONFIG_LWIP_IP4_REASSEMBLY

Enable reassembly incoming fragmented IP4 packets

*Found in: Component config > LWIP*

Enabling this option allows reassembling incoming fragmented IP4 packets.

**Default value:**
- No (disabled)

CONFIG_LWIP_IP6_REASSEMBLY

Enable reassembly incoming fragmented IP6 packets

*Found in: Component config > LWIP*

Enabling this option allows reassembling incoming fragmented IP6 packets.

**Default value:**
- No (disabled)

CONFIG_LWIP_IP_FORWARD

Enable IP forwarding

*Found in: Component config > LWIP*

Enabling this option allows packets forwarding across multiple interfaces.

**Default value:**
- No (disabled)
**CONFIG_LWIP_IPV4_NAPT**

Enable NAT (new/experimental)

*Found in:* Component config > LWIP > CONFIG_LWIP_IP_FORWARD

Enabling this option allows Network Address and Port Translation.

**Default value:**
- No (disabled) if CONFIG_LWIP_IP_FORWARD

**CONFIG_LWIP_STATS**

Enable LWIP statistics

*Found in:* Component config > LWIP

Enabling this option allows LWIP statistics

**Default value:**
- No (disabled)

**CONFIG_LWIP_ESP_GRATUITOUS_ARP**

Send gratuitous ARP periodically

*Found in:* Component config > LWIP

Enable this option allows to send gratuitous ARP periodically.

This option solve the compatibility issues. If the ARP table of the AP is old, and the AP doesn’t send ARP request to update it’s ARP table, this will lead to the STA sending IP packet fail. Thus we send gratuitous ARP periodically to let AP update it’s ARP table.

**Default value:**
- Yes (enabled)

**CONFIG_LWIP_GARP_TMR_INTERVAL**

GARP timer interval(seconds)

*Found in:* Component config > LWIP > CONFIG_LWIP_ESP_GRATUITOUS_ARP

Set the timer interval for gratuitous ARP. The default value is 60s

**Default value:**
- 60

**CONFIG_LWIP_TCPIP_RECVMBOX_SIZE**

TCPIP task receive mail box size

*Found in:* Component config > LWIP

Set TCPIP task receive mail box size. Generally bigger value means higher throughput but more memory. The value should be bigger than UDP/TCP mail box size.

**Range:**
- from 6 to 64 if CONFIG_LWIP_WND_SCALE
- from 6 to 1024 if CONFIG_LWIP_WND_SCALE

**Default value:**
- 32
CONFIG_LWIP_DHCP_DOES_ARP_CHECK

DHCP: Perform ARP check on any offered address

*Found in: Component config > LWIP*

Enabling this option performs a check (via ARP request) if the offered IP address is not already in use by another host on the network.

**Default value:**
- Yes (enabled)

CONFIG_LWIP_DHCP_DISABLE_CLIENT_ID

DHCP: Disable Use of HW address as client identification

*Found in: Component config > LWIP*

This option could be used to disable DHCP client identification with its MAC address. (Client id is used by DHCP servers to uniquely identify clients and are included in the DHCP packets as an option 61) Set this option to “y” in order to exclude option 61 from DHCP packets.

**Default value:**
- No (disabled)

CONFIG_LWIP_DHCP_DISABLE_VENDOR_CLASS_ID

DHCP: Disable Use of vendor class identification

*Found in: Component config > LWIP*

This option could be used to disable DHCP client vendor class identification. Set this option to “y” in order to exclude option 60 from DHCP packets.

**Default value:**
- Yes (enabled)

CONFIG_LWIP_DHCP_RESTORE_LAST_IP

DHCP: Restore last IP obtained from DHCP server

*Found in: Component config > LWIP*

When this option is enabled, DHCP client tries to re-obtain last valid IP address obtained from DHCP server. Last valid DHCP configuration is stored in nvs and restored after reset/power-up. If IP is still available, there is no need for sending discovery message to DHCP server and save some time.

**Default value:**
- No (disabled)

CONFIG_LWIP_DHCP_OPTIONS_LEN

DHCP total option length

*Found in: Component config > LWIP*

Set total length of outgoing DHCP option msg. Generally bigger value means it can carry more options and values. If your code meets LWIP_ASSERT due to option value is too long. Please increase the LWIP_DHCP_OPTIONS_LEN value.

**Range:**
- from 68 to 255

**Default value:**
- 68
- 108
**CONFIG_LWIP_NUM_NETIF_CLIENT_DATA**

Number of clients store data in netif

*Found in: Component config > LWIP*

Number of clients that may store data in client_data member array of struct netif.

**Range:**
- from 0 to 256

**Default value:**
- 0

**DHCP server** Contains:
- **CONFIG_LWIP_DHCPS**

**CONFIG_LWIP_DHCPS**

DHCPs: Enable IPv4 Dynamic Host Configuration Protocol Server (DHCPs)

*Found in: Component config > LWIP > DHCP server*

Enabling this option allows the device to run the DHCP server (to dynamically assign IPv4 addresses to clients).

**Default value:**
- Yes (enabled)

**CONFIG_LWIP_DHCPS_LEASE_UNIT**

Multiplier for lease time, in seconds

*Found in: Component config > LWIP > DHCP server > CONFIG_LWIP_DHCPS*

The DHCP server is calculating lease time multiplying the sent and received times by this number of seconds per unit. The default is 60, that equals one minute.

**Range:**
- from 1 to 3600

**Default value:**
- 60

**CONFIG_LWIP_DHCPS_MAX_STATION_NUM**

Maximum number of stations

*Found in: Component config > LWIP > DHCP server > CONFIG_LWIP_DHCPS*

The maximum number of DHCP clients that are connected to the server. After this number is exceeded, DHCP server removes of the oldest device from it’s address pool, without notification.

**Range:**
- from 1 to 64

**Default value:**
- 8

**CONFIG_LWIP_AUTOIP**

Enable IPV4 Link-Local Addressing (AUTOIP)

*Found in: Component config > LWIP*

Enabling this option allows the device to self-assign an address in the 169.256/16 range if none is assigned statically or via DHCP.
See RFC 3927.

**Default value:**
- No (disabled)

**Contains:**
- `CONFIG_LWIP_AUTOIP_TRIES`
- `CONFIG_LWIP_AUTOIP_MAX_CONFLICTS`
- `CONFIG_LWIP_AUTOIP_RATE_LIMIT_INTERVAL`

**CONFIG_LWIP_AUTOIP_TRIES**

DHCP Probes before self-assigning IPv4 LL address

*Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP*

DHCP client will send this many probes before self-assigning a link local address.

From LWIP help: “This can be set as low as 1 to get an AutoIP address very quickly, but you should be prepared to handle a changing IP address when DHCP overrides AutoIP.” (In the case of ESP-IDF, this means multiple SYSTEM_EVENT_STA_GOT_IP events.)

**Range:**
- from 1 to 100 if `CONFIG_LWIP_AUTOIP`

**Default value:**
- 2 if `CONFIG_LWIP_AUTOIP`

**CONFIG_LWIP_AUTOIP_MAX_CONFLICTS**

Max IP conflicts before rate limiting

*Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP*

If the AUTOIP functionality detects this many IP conflicts while self-assigning an address, it will go into a rate limited mode.

**Range:**
- from 1 to 100 if `CONFIG_LWIP_AUTOIP`

**Default value:**
- 9 if `CONFIG_LWIP_AUTOIP`

**CONFIG_LWIP_AUTOIP_RATE_LIMIT_INTERVAL**

Rate limited interval (seconds)

*Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP*

If rate limiting self-assignment requests, wait this long between each request.

**Range:**
- from 5 to 120 if `CONFIG_LWIP_AUTOIP`

**Default value:**
- 20 if `CONFIG_LWIP_AUTOIP`

**CONFIG_LWIP_IPV6**

Enable IPv6

*Found in: Component config > LWIP*

Enable IPv6 function. If not use IPv6 function, set this option to n. If disabling LWIP_IPV6 then some other components (coap and asio) will no longer be available.

**Default value:**
Chapter 2. API

**CONFIG_LWIP_IPV6_AUTOCONFIG**

Enable IPv6 stateless address autoconfiguration (SLAAC)

*Found in: Component config > LWIP > CONFIG_LWIP_IPV6*

Enabling this option allows the devices to IPv6 stateless address autoconfiguration (SLAAC).

Default value:

- No (disabled)

**CONFIG_LWIP_IPV6_NUM_ADDRESSES**

Number of IPv6 addresses on each network interface

*Found in: Component config > LWIP > CONFIG_LWIP_IPV6*

The maximum number of IPv6 addresses on each interface. Any additional addresses will be discarded.

Default value:

- 3

**CONFIG_LWIP_IPV6_FORWARD**

Enable IPv6 forwarding between interfaces

*Found in: Component config > LWIP > CONFIG_LWIP_IPV6*

Forwarding IPv6 packets between interfaces is only required when acting as a router.

Default value:

- No (disabled)

**CONFIG_LWIP_IPV6_RDNSS_MAX_DNS_SERVERS**

Use IPv6 Router Advertisement Recursive DNS Server Option

*Found in: Component config > LWIP*

Use IPv6 Router Advertisement Recursive DNS Server Option (as per RFC 6106) to copy a defined maximum number of DNS servers to the DNS module. Set this option to a number of desired DNS servers advertised in the RA protocol. This feature is disabled when set to 0.

Default value:

- 0 if `CONFIG_LWIP_IPV6_AUTOCONFIG`

**CONFIG_LWIP_IPV6_DHCP6**

Enable DHCPv6 stateless address autoconfiguration

*Found in: Component config > LWIP*

Enable DHCPv6 for IPv6 stateless address autoconfiguration. Note that the dhcpv6 client has to be started using dhcp6_enable_stateless(netif); Note that the stateful address autoconfiguration is not supported.

Default value:

- No (disabled) if `CONFIG_LWIP_IPV6_AUTOCONFIG`
**CONFIG_LWIP_NETIF_STATUS_CALLBACK**

Enable status callback for network interfaces

*Found in: Component config > LWIP*

Enable callbacks when the network interface is up/down and addresses are changed.

**Default value:**
- No (disabled)

**CONFIG_LWIP_NETIF_LOOPBACK**

Support per-interface loopback

*Found in: Component config > LWIP*

Enabling this option means that if a packet is sent with a destination address equal to the interface’s own IP address, it will “loop back” and be received by this interface. Disabling this option disables support of loopback interface in lwIP

**Default value:**
- Yes (enabled)

Contains:

- `CONFIG_LWIP_LOOPBACK_MAX_PBUFS`

**CONFIG_LWIP_LOOPBACK_MAX_PBUFS**

Max queued loopback packets per interface

*Found in: Component config > LWIP > CONFIG_LWIP_NETIF_LOOPBACK*

Configure the maximum number of packets which can be queued for loopback on a given interface. Reducing this number may cause packets to be dropped, but will avoid filling memory with queued packet data.

**Range:**
- from 0 to 16

**Default value:**
- 8

**TCP**

Contains:

- `CONFIG_LWIP_TCP_WND_DEFAULT`
- `CONFIG_LWIP_TCP_SND_BUF_DEFAULT`
- `CONFIG_LWIP_TCP_RECVMBX_SIZE`
- `CONFIG_LWIP_TCP_RTO_TIME`
- `CONFIG_LWIP_MAX_ACTIVE_TCP`
- `CONFIG_LWIP_TCP_FIN_WAIT_TIMEOUT`
- `CONFIG_LWIP_MAX_LISTENING_TCP`
- `CONFIG_LWIP_TCP_MAXRTX`
- `CONFIG_LWIP_TCP_SYNMAXRTX`
- `CONFIG_LWIP_TCP_MSL`
- `CONFIG_LWIP_TCP_MSS`
- `CONFIG_LWIP_TCP_OVERSIZE`
- `CONFIG_LWIP_TCP_QUEUE_OOSEQ`
- `CONFIG_LWIP_WND_SCALE`
- `CONFIG_LWIP_TCP_HIGH_SPEED_RETRANSMISSION`
- `CONFIG_LWIP_TCP_TMR_INTERVAL`
**CONFIG_LWIP_MAX_ACTIVE_TCP**

Maximum active TCP Connections

*Found in: Component config > LWIP > TCP*

The maximum number of simultaneously active TCP connections. The practical maximum limit is determined by available heap memory at runtime.

Changing this value by itself does not substantially change the memory usage of LWIP, except for preventing new TCP connections after the limit is reached.

*Range:*

  • from 1 to 1024

*Default value:*

  • 16

**CONFIG_LWIP_MAX_LISTENING_TCP**

Maximum listening TCP Connections

*Found in: Component config > LWIP > TCP*

The maximum number of simultaneously listening TCP connections. The practical maximum limit is determined by available heap memory at runtime.

Changing this value by itself does not substantially change the memory usage of LWIP, except for preventing new listening TCP connections after the limit is reached.

*Range:*

  • from 1 to 1024

*Default value:*

  • 16

**CONFIG_LWIP_TCP_HIGH_SPEED_RETRANSMISSION**

TCP high speed retransmissions

*Found in: Component config > LWIP > TCP*

Speed up the TCP retransmission interval. If disabled, it is recommended to change the number of SYN retransmissions to 6, and TCP initial rto time to 3000.

*Default value:*

  • Yes (enabled)

**CONFIG_LWIP_TCP_MAXRTX**

Maximum number of retransmissions of data segments

*Found in: Component config > LWIP > TCP*

Set maximum number of retransmissions of data segments.

*Range:*

  • from 3 to 12

*Default value:*

  • 12

**CONFIG_LWIP_TCP_SYNMAXRTX**

Maximum number of retransmissions of SYN segments

*Found in: Component config > LWIP > TCP*

Set maximum number of retransmissions of SYN segments.
Range:  
- from 3 to 12  
Default value:  
- 6  
- 12

**CONFIG_LWIP_TCP_MSS**

Maximum Segment Size (MSS)

*Found in: Component config > LWIP > TCP*

Set maximum segment size for TCP transmission.

Can be set lower to save RAM, the default value 1460(ipv4)/1440(ipv6) will give best throughput. IPv4 TCP_MSS Range: 576 <= TCP_MSS <= 1460 IPv6 TCP_MSS Range: 1220 <= TCP_mSS <= 1440

Range:  
- from 536 to 1460  
Default value:  
- 1440

**CONFIG_LWIP_TCP_TMR_INTERVAL**

TCP timer interval(ms)

*Found in: Component config > LWIP > TCP*

Set TCP timer interval in milliseconds.

Can be used to speed connections on bad networks. A lower value will redeliver unacked packets faster.

Default value:  
- 250

**CONFIG_LWIP_TCP_MSL**

Maximum segment lifetime (MSL)

*Found in: Component config > LWIP > TCP*

Set maximum segment lifetime in milliseconds.

Default value:  
- 60000

**CONFIG_LWIP_TCP_FIN_WAIT_TIMEOUT**

Maximum FIN segment lifetime

*Found in: Component config > LWIP > TCP*

Set maximum segment lifetime in milliseconds.

Default value:  
- 20000

**CONFIG_LWIP_TCP_SND_BUF_DEFAULT**

Default send buffer size

*Found in: Component config > LWIP > TCP*

Set default send buffer size for new TCP sockets.

Per-socket send buffer size can be changed at runtime with lwip_setsockopt(s, TCP_SNDBUF, …).
This value must be at least 2x the MSS size, and the default is 4x the default MSS size.

Setting a smaller default SNDBUF size can save some RAM, but will decrease performance.

**Range:**
- from 2440 to 65535 if `CONFIG_LWIP_WND_SCALE`
- from 2440 to 1024000 if `CONFIG_LWIP_WND_SCALE`

**Default value:**
- 5744

**CONFIG_LWIP_TCP_WND_DEFAULT**

Default receive window size

*Found in: Component config > LWIP > TCP*

Set default TCP receive window size for new TCP sockets.

Per-socket receive window size can be changed at runtime with lwip_setsockopt(s, TCP_WINDOW, …).

Setting a smaller default receive window size can save some RAM, but will significantly decrease performance.

**Range:**
- from 2440 to 65535 if `CONFIG_LWIP_WND_SCALE`
- from 2440 to 1024000 if `CONFIG_LWIP_WND_SCALE`

**Default value:**
- 5744

**CONFIG_LWIP_TCP_RECCVMBOX_SIZE**

Default TCP receive mail box size

*Found in: Component config > LWIP > TCP*

Set TCP receive mail box size. Generally bigger value means higher throughput but more memory. The recommended value is: \( \frac{LWIP_TCP_WND_DEFAULT}{TCP_MSS} + 2 \). e.g. if \( LWIP_TCP_WND_DEFAULT=14360 \), \( TCP_MSS=1436 \), then the recommended receive mail box size is \( \frac{14360}{1436} + 2 = 12 \).

TCP receive mail box is a per socket mail box, when the application receives packets from TCP socket, LWIP core firstly posts the packets to TCP receive mail box and the application then fetches the packets from mail box. It means LWIP can caches maximum LWIP_TCP_RECCVMBOX_SIZE packets for each TCP socket, so the maximum possible cached TCP packets for all TCP sockets is LWIP_TCP_RECCVMBOX_SIZE multiples the maximum TCP socket number. In other words, the bigger LWIP_TCP_RECCVMBOX_SIZE means more memory. On the other hand, if the receive mail box is too small, the mail box may be full. If the mail box is full, the LWIP drops the packets. So generally we need to make sure the TCP receive mail box is big enough to avoid packet drop between LWIP core and application.

**Range:**
- from 6 to 64 if `CONFIG_LWIP_WND_SCALE`
- from 6 to 1024 if `CONFIG_LWIP_WND_SCALE`

**Default value:**
- 6

**CONFIG_LWIP_TCP_QUEUE_OOSEQ**

Queue incoming out-of-order segments

*Found in: Component config > LWIP > TCP*

Queue incoming out-of-order segments for later use.
Disable this option to save some RAM during TCP sessions, at the expense of increased retransmissions if segments arrive out of order.

**Default value:**
- Yes (enabled)

**CONFIG_LWIP_TCP_SACK_OUT**
Support sending selective acknowledgements

*Found in: Component config > LWIP > TCP > CONFIG_LWIP_TCP_QUEUE_OOSEQ*

TCP will support sending selective acknowledgements (SACKs).

**Default value:**
- No (disabled)

**CONFIG_LWIP_TCP_OVERSIZE**
Pre-allocate transmit PBUF size

*Found in: Component config > LWIP > TCP*

Allows enabling “oversize” allocation of TCP transmission pbufs ahead of time, which can reduce the length of pbuf chains used for transmission.

This will not make a difference to sockets where Nagle’s algorithm is disabled.

Default value of MSS is fine for most applications, 25% MSS may save some RAM when only transmitting small amounts of data. Disabled will have worst performance and fragmentation characteristics, but uses least RAM overall.

**Available options:**
- MSS (LWIP_TCP_OVERSIZE_MSS)
- 25% MSS (LWIP_TCP_OVERSIZE_QUARTER_MSS)
- Disabled (LWIP_TCP_OVERSIZE_DISABLE)

**CONFIG_LWIP_WND_SCALE**
Support TCP window scale

*Found in: Component config > LWIP > TCP*

Enable this feature to support TCP window scaling.

**Default value:**
- No (disabled) if CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP

**CONFIG_LWIP_TCP_RCV_SCALE**
Set TCP receiving window scaling factor

*Found in: Component config > LWIP > TCP > CONFIG_LWIP_WND_SCALE*

Enable this feature to support TCP window scaling.

**Range:**
- from 0 to 14 if CONFIG_LWIP_WND_SCALE

**Default value:**
- 0 if CONFIG_LWIP_WND_SCALE
### Chapter 2. API 参考

#### CONFIG_LWIP_TCP_RTO_TIME

Default TCP rto time

*Found in: Component config > LWIP > TCP*

Set default TCP rto time for a reasonable initial rto. In bad network environment, recommend set value of rto time to 1500.

*Default value:*
- 3000
- 1500

#### UDP

Contains:
- **CONFIG_LWIP_UDP_RECVMBOX_SIZE**
- **CONFIG_LWIP_MAX_UDP_PCBS**

#### CONFIG_LWIP_MAX_UDP_PCBS

Maximum active UDP control blocks

*Found in: Component config > LWIP > UDP*

The maximum number of active UDP “connections” (ie UDP sockets sending/receiving data). The practical maximum limit is determined by available heap memory at runtime.

*Range:*
- from 1 to 1024

*Default value:*
- 16

#### CONFIG_LWIP_UDP_RECVMBOX_SIZE

Default UDP receive mail box size

*Found in: Component config > LWIP > UDP*

Set UDP receive mail box size. The recommended value is 6.

UDP receive mail box is a per socket mail box, when the application receives packets from UDP socket, LWIP core firstly posts the packets to UDP receive mail box and the application then fetches the packets from mail box. It means LWIP can caches maximum UDP_RECCVMBOX_SIZE packets for each UDP socket, so the maximum possible cached UDP packets for all UDP sockets is UDP_RECCVMBOX_SIZE multiples the maximum UDP socket number. In other words, the bigger UDP_RECCVMBOX_SIZE means more memory. On the other hand, if the receiv mail box is too small, the mail box may be full. If the mail box is full, the LWIP drops the packets. So generally we need to make sure the UDP receive mail box is big enough to avoid packet drop between LWIP core and application.

*Range:*
- from 6 to 64

*Default value:*
- 6

#### Checksums

Contains:
- **CONFIG_LWIP_CHECKSUM_CHECK_ICMP**
- **CONFIG_LWIP_CHECKSUM_CHECK_IP**
- **CONFIG_LWIP_CHECKSUM_CHECK_UDP**
Chapter 2. API 参考

**CONFIG_LWIP_CHECKSUM_CHECK_IP**
Enable LWIP IP checksums

*Found in: Component config > LWIP > Checksums*
Enable checksum checking for received IP messages

**Default value:**
- No (disabled)

**CONFIG_LWIP_CHECKSUM_CHECK_UDP**
Enable LWIP UDP checksums

*Found in: Component config > LWIP > Checksums*
Enable checksum checking for received UDP messages

**Default value:**
- No (disabled)

**CONFIG_LWIP_CHECKSUM_CHECK_ICMP**
Enable LWIP ICMP checksums

*Found in: Component config > LWIP > Checksums*
Enable checksum checking for received ICMP messages

**Default value:**
- Yes (enabled)

**CONFIG_LWIP_TCPIP_TASK_STACK_SIZE**
TCP/IP Task Stack Size

*Found in: Component config > LWIP*
Configure TCP/IP task stack size, used by LWIP to process multi-threaded TCP/IP operations. Setting this stack too small will result in stack overflow crashes.

**Range:**
- from 2048 to 65536

**Default value:**
- 3072

**CONFIG_LWIP_TCPIP_TASK_AFFINITY**
TCP/IP task affinity

*Found in: Component config > LWIP*
Allows setting LwIP tasks affinity, i.e. whether the task is pinned to CPU0, pinned to CPU1, or allowed to run on any CPU. Currently this applies to “TCP/IP” task and “Ping” task.

**Available options:**
- No affinity (LWIP_TCPIP_TASK_AFFINITY_NO_AFFINITY)
- CPU0 (LWIP_TCPIP_TASK_AFFINITY_CPU0)
- CPU1 (LWIP_TCPIP_TASK_AFFINITY_CPU1)
**CONFIG_LWIP_PPP_SUPPORT**

Enable PPP support (new/experimental)

*Found in: Component config > LWIP*

Enable PPP stack. Now only PPP over serial is possible. PPP over serial support is experimental and unsupported.

**Default value:**
- No (disabled)

*Contains:*
- `CONFIG_LWIP_PPP_ENABLE_IPV6`

**CONFIG_LWIP_PPP_ENABLE_IPV6**

Enable IPv6 support for PPP connections (IPv6CP)

*Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT*

Enable IPv6 support in PPP for the local link between the DTE (processor) and DCE (modem). There are some modems which do not support the IPv6 addressing in the local link. If they are requested for IPv6CP negotiation, they may time out. This would in turn fail the configuration for the whole link. If your modem is not responding correctly to PPP Phase Network, try to disable IPv6 support.

**Default value:**
- Yes (enabled) if `CONFIG_LWIP_PPP_SUPPORT` && `CONFIG_LWIP_IPV6`

**CONFIG_LWIP_IPV6_MEMP_NUM_ND6_QUEUE**

Max number of IPv6 packets to queue during MAC resolution

*Found in: Component config > LWIP*

Config max number of IPv6 packets to queue during MAC resolution.

**Range:**
- from 3 to 20

**Default value:**
- 3

**CONFIG_LWIP_IPV6_NUM_ND6_NUM_NEIGHBORS**

Max number of entries in IPv6 neighbor cache

*Found in: Component config > LWIP*

Config max number of entries in IPv6 neighbor cache

**Range:**
- from 3 to 10

**Default value:**
- 5

**CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT**

Enable Notify Phase Callback

*Found in: Component config > LWIP*

Enable to set a callback which is called on change of the internal PPP state machine.

**Default value:**
- No (disabled) if `CONFIG_LWIP_PPP_SUPPORT`
CONFIG_LWIP_PPP_PAP_SUPPORT
Enable PAP support

Found in: Component config > LWIP
Enable Password Authentication Protocol (PAP) support

Default value:
• No (disabled) if CONFIG_LWIP_PPP_SUPPORT

CONFIG_LWIP_PPP_CHAP_SUPPORT
Enable CHAP support

Found in: Component config > LWIP
Enable Challenge Handshake Authentication Protocol (CHAP) support

Default value:
• No (disabled) if CONFIG_LWIP_PPP_SUPPORT

CONFIG_LWIP_PPP_MSCHAP_SUPPORT
Enable MSCHAP support

Found in: Component config > LWIP
Enable Microsoft version of the Challenge-Handshake Authentication Protocol (MSCHAP) support

Default value:
• No (disabled) if CONFIG_LWIP_PPP_SUPPORT

CONFIG_LWIP_PPP_MPPE_SUPPORT
Enable MPPE support

Found in: Component config > LWIP
Enable Microsoft Point-to-Point Encryption (MPPE) support

Default value:
• No (disabled) if CONFIG_LWIP_PPP_SUPPORT

CONFIG_LWIP_ENABLE_LCP_ECHO
Enable LCP ECHO

Found in: Component config > LWIP
Enable LCP echo keepalive requests

Default value:
• No (disabled) if CONFIG_LWIP_PPP_SUPPORT

CONFIG_LWIP_LCP_ECHO_INTERVAL
Echo interval (s)

Found in: Component config > LWIP > CONFIG_LWIP_ENABLE_LCP_ECHO
Interval in seconds between keepalive LCP echo requests, 0 to disable.

Range:
• from 0 to 1000000 if CONFIG_LWIP_ENABLE_LCP_ECHO

Default value:
• 3 if CONFIG_LWIP_ENABLE_LCP_ECHO
CONFIG_LWIP_LCP_MAXECHOFAILS

Maximum echo failures

*Found in: Component config > LWIP > CONFIG_LWIP_ENABLE_LCP_ECHO*

Number of consecutive unanswered echo requests before failure is indicated.

**Range:**
- from 0 to 100000 if `CONFIG_LWIP_ENABLE_LCP_ECHO`

**Default value:**
- 3 if `CONFIG_LWIP_ENABLE_LCP_ECHO`

CONFIG_LWIP_PPP_DEBUG_ON

Enable PPP debug log output

*Found in: Component config > LWIP*

Enable PPP debug log output

**Default value:**
- No (disabled) if `CONFIG_LWIP_PPP_SUPPORT`

CONFIG_LWIP_SLIP_SUPPORT

Enable SLIP support (new/experimental)

*Found in: Component config > LWIP*

Enable SLIP stack. Now only SLIP over serial is possible.

SLIP over serial support is experimental and unsupported.

**Default value:**
- No (disabled)

Contains:
- `CONFIG_LWIP_SLIP_DEBUG_ON`

CONFIG_LWIP_SLIP_DEBUG_ON

Enable SLIP debug log output

*Found in: Component config > LWIP > CONFIG_LWIP_SLIP_SUPPORT*

Enable SLIP debug log output

**Default value:**
- No (disabled) if `CONFIG_LWIP_SLIP_SUPPORT`

ICMP

Contains:
- `CONFIG_LWIP_ICMP`
- `CONFIG_LWIP_BROADCAST_PING`
- `CONFIG_LWIP_MULTICAST_PING`

CONFIG_LWIP_ICMP

ICMP: Enable ICMP

*Found in: Component config > LWIP > ICMP*

Enable ICMP module for check network stability

**Default value:**
Chapter 2. API

- Yes (enabled)

**CONFIG_LWIP_MULTICAST_PING**

Respond to multicast pings

*Found in: Component config > LWIP > ICMP*

*Default value:*
- No (disabled)

**CONFIG_LWIP_BROADCAST_PING**

Respond to broadcast pings

*Found in: Component config > LWIP > ICMP*

*Default value:*
- No (disabled)

**LWIP RAW API**  Contains:
- **CONFIG_LWIP_MAX_RAW_PCBS**

**CONFIG_LWIP_MAX_RAW_PCBS**

Maximum LWIP RAW PCBs

*Found in: Component config > LWIP > LWIP RAW API*

The maximum number of simultaneously active LWIP RAW protocol control blocks. The practical maximum limit is determined by available heap memory at runtime.

*Range:*
- from 1 to 1024

*Default value:*
- 16

**SNTP**  Contains:
- **CONFIG_LWIP_SNTP_MAX_SERVERS**
- **CONFIG_LWIP_SNTP_UPDATE_DELAY**
- **CONFIG_LWIP_DHCP_GET_NTP_SRV**

**CONFIG_LWIP_SNTP_MAX_SERVERS**

Maximum number of NTP servers

*Found in: Component config > LWIP > SNTP*

Set maximum number of NTP servers used by LwIP SNTP module. First argument of snntp_setserver/snntp_setservername functions is limited to this value.

*Range:*
- from 1 to 16

*Default value:*
- 1
**CONFIG_LWIP_DHCP_GET_NTP_SRV**

Request NTP servers from DHCP

*Found in: Component config > LWIP > SNTP*

If enabled, LWIP will add ‘NTP’ to Parameter-Request Option sent via DHCP-request. DHCP server might reply with an NTP server address in option 42. SNTP callback for such replies should be set accordingly (see snntp_servermode_dhcp() func.)

**Default value:**
- No (disabled)

**CONFIG_LWIP_DHCP_MAX_NTP_SERVERS**

Maximum number of NTP servers acquired via DHCP

*Found in: Component config > LWIP > SNTP > CONFIG_LWIP_DHCP_GET_NTP_SRV*

Set maximum number of NTP servers acquired via DHCP-offer. Should be less or equal to “Maximum number of NTP servers”, any extra servers would be just ignored.

**Range:**
- from 1 to 16 if **CONFIG_LWIP_DHCP_GET_NTP_SRV**

**Default value:**
- 1 if **CONFIG_LWIP_DHCP_GET_NTP_SRV**

**CONFIG_LWIP_SNP_UPDATE_DELAY**

Request interval to update time (ms)

*Found in: Component config > LWIP > SNTP*

This option allows you to set the time update period via SNTP. Default is 1 hour. Must not be below 15 seconds by specification. (SNTPv4 RFC 4330 enforces a minimum update time of 15 seconds).

**Range:**
- from 15000 to 4294967295

**Default value:**
- 3600000

**CONFIG_LWIP_BRIDGEIF_MAX_PORTS**

Maximum number of bridge ports

*Found in: Component config > LWIP*

Set maximum number of ports a bridge can consist of.

**Range:**
- from 1 to 63

**Default value:**
- 7

**CONFIG_LWIP_ESP_LWIP_ASSERT**

Enable LWIP ASSERT checks

*Found in: Component config > LWIP*

Enable this option keeps LWIP assertion checks enabled. It is recommended to keep this option enabled. If asserts are disabled for the entire project, they are also disabled for LWIP and this option is ignored.

**Default value:**
- Yes (enabled) if COMPILER_OPTIMIZATION_ASSERTIONS_DISABLE
Hooks Contains:

- `CONFIG_LWIP_HOOK_ND6_GET_GW`
- `CONFIG_LWIP_HOOK_IP6_INPUT`
- `CONFIG_LWIP_HOOK_IP6_ROUTE`
- `CONFIG_LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE`
- `CONFIG_LWIP_HOOK_TCP_ISN`

**CONFIG_LWIP_HOOK_TCP_ISN**

TCP ISN Hook

*Found in: Component config > LWIP > Hooks*

Enables to define a TCP ISN hook to randomize initial sequence number in TCP connection. The default TCP ISN algorithm used in IDF (standardized in RFC 6528) produces ISN by combining an MD5 of the new TCP id and a stable secret with the current time. This is because the lwIP implementation (`tcp_next_iss`) is not very strong, as it does not take into consideration any platform specific entropy source.

Set to `LWIP_HOOK_TCP_ISN_CUSTOM` to provide custom implementation. Set to `LWIP_HOOK_TCP_ISN_NONE` to use lwIP implementation.

Available options:

- No hook declared (`LWIP_HOOK_TCP_ISN_NONE`)
- Default implementation (`LWIP_HOOK_TCP_ISN_DEFAULT`)
- Custom implementation (`LWIP_HOOK_TCP_ISN_CUSTOM`)

**CONFIG_LWIP_HOOK_IP6_ROUTE**

IPv6 route Hook

*Found in: Component config > LWIP > Hooks*

Enables custom IPv6 route hook. Setting this to “default” provides weak implementation stub that could be overwritten in application code. Setting this to “custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (`LWIP_HOOK_IP6_ROUTE_NONE`)
- Default (weak) implementation (`LWIP_HOOK_IP6_ROUTE_DEFAULT`)
- Custom implementation (`LWIP_HOOK_IP6_ROUTE_CUSTOM`)

**CONFIG_LWIP_HOOK_ND6_GET_GW**

IPv6 get gateway Hook

*Found in: Component config > LWIP > Hooks*

Enables custom IPv6 route hook. Setting this to “default” provides weak implementation stub that could be overwritten in application code. Setting this to “custom” provides hook’s declaration only and expects the application to implement it.

Available options:

- No hook declared (`LWIP_HOOK_ND6_GET_GW_NONE`)
- Default (weak) implementation (`LWIP_HOOK_ND6_GET_GW_DEFAULT`)
- Custom implementation (`LWIP_HOOK_ND6_GET_GW_CUSTOM`)

**CONFIG_LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE**

Netconn external resolve Hook

*Found in: Component config > LWIP > Hooks*
Enables custom DNS resolve hook. Setting this to “default” provides weak implementation stub that could be overwritten in application code. Setting this to “custom” provides hook’s declaration only and expects the application to implement it.

**Available options:***
- No hook declared (LWIP_HOOK_NETCONN_EXT_RESOLVE_NONE)
- Default (weak) implementation (LWIP_HOOK_NETCONN_EXT_RESOLVE_DEFAULT)
- Custom implementation (LWIP_HOOK_NETCONN_EXT_RESOLVE_CUSTOM)

**CONFIG_LWIP_HOOK_IP6_INPUT**

IPv6 packet input

*Found in: Component config > LWIP > Hooks*

Enables custom IPv6 packet input. Setting this to “default” provides weak implementation stub that could be overwritten in application code. Setting this to “custom” provides hook’s declaration only and expects the application to implement it.

**Available options:***
- No hook declared (LWIP_HOOK_IP6_INPUT_NONE)
- Default (weak) implementation (LWIP_HOOK_IP6_INPUT_DEFAULT)
- Custom implementation (LWIP_HOOK_IP6_INPUT_CUSTOM)

**CONFIG_LWIP_DEBUG**

Enable LWIP Debug

*Found in: Component config > LWIP*

Enabling this option allows different kinds of lwIP debug output.

All lwIP debug features increase the size of the final binary.

**Default value:**
- No (disabled)

Contains:
- `CONFIG_LWIP_API_LIB_DEBUG`
- `CONFIG_LWIP_BRIDGEIF_FDB_DEBUG`
- `CONFIG_LWIP_BRIDGEIF_FW_DEBUG`
- `CONFIG_LWIP_BRIDGEIF_DEBUG`
- `CONFIG_LWIP_DHCP_DEBUG`
- `CONFIG_LWIP_DHCP_STATE_DEBUG`
- `CONFIG_LWIP_DNS_DEBUG`
- `CONFIG_LWIP_ETHARP_DEBUG`
- `CONFIG_LWIP_ICMP_DEBUG`
- `CONFIG_LWIP_ICMP6_DEBUG`
- `CONFIG_LWIP_IP_DEBUG`
- `CONFIG_LWIP_IP6_DEBUG`
- `CONFIG_LWIP_NETIF_DEBUG`
- `CONFIG_LWIP_PBUF_DEBUG`
- `CONFIG_LWIP_Sntp_DEBUG`
- `CONFIG_LWIP_SOCKETS_DEBUG`
- `CONFIG_LWIP_TCP_DEBUG`
- `CONFIG_LWIP_DEBUG_ESP_LOG`

**CONFIG_LWIP_DEBUG_ESP_LOG**

Route LWIP debugs through ESP_LOG interface

*Found in: Component config > LWIP > CONFIG_LWIP_DEBUG*
Enabling this option routes all enabled LWIP debugs through ESP_LOGD.

**Default value:**
- No (disabled) if `CONFIG_LWIP_DEBUG`

**CONFIG_LWIP_NETIF_DEBUG**

Enable netif debug messages

*Found in: Component config > LWIP > CONFIG_LWIP_DEBUG*

**Default value:**
- No (disabled) if `CONFIG_LWIP_DEBUG`

**CONFIG_LWIP_PBUF_DEBUG**

Enable pbuf debug messages

*Found in: Component config > LWIP > CONFIG_LWIP_DEBUG*

**Default value:**
- No (disabled) if `CONFIG_LWIP_DEBUG`

**CONFIG_LWIP_ETHARP_DEBUG**

Enable etharp debug messages

*Found in: Component config > LWIP > CONFIG_LWIP_DEBUG*

**Default value:**
- No (disabled) if `CONFIG_LWIP_DEBUG`

**CONFIG_LWIP_API_LIB_DEBUG**

Enable api lib debug messages

*Found in: Component config > LWIP > CONFIG_LWIP_DEBUG*

**Default value:**
- No (disabled) if `CONFIG_LWIP_DEBUG`

**CONFIG_LWIP_SOCKETS_DEBUG**

Enable socket debug messages

*Found in: Component config > LWIP > CONFIG_LWIP_DEBUG*

**Default value:**
- No (disabled) if `CONFIG_LWIP_DEBUG`

**CONFIG_LWIP_IP_DEBUG**

Enable IP debug messages

*Found in: Component config > LWIP > CONFIG_LWIP_DEBUG*

**Default value:**
- No (disabled) if `CONFIG_LWIP_DEBUG`
CONFIG_LWIP_ICMP_DEBUG
Enable ICMP debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:*
• No (disabled) if CONFIG_LWIP_DEBUG && CONFIG_LWIP_ICMP

CONFIG_LWIP_DHCP_STATE_DEBUG
Enable DHCP state tracking

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:*
• No (disabled) if CONFIG_LWIP_DEBUG

CONFIG_LWIP_DHCP_DEBUG
Enable DHCP debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:*
• No (disabled) if CONFIG_LWIP_DEBUG

CONFIG_LWIP_IP6_DEBUG
Enable IP6 debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:*
• No (disabled) if CONFIG_LWIP_DEBUG

CONFIG_LWIP_ICMP6_DEBUG
Enable ICMP6 debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:*
• No (disabled) if CONFIG_LWIP_DEBUG

CONFIG_LWIP_TCP_DEBUG
Enable TCP debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:*
• No (disabled) if CONFIG_LWIP_DEBUG

CONFIG_LWIP_SNTP_DEBUG
Enable SNTP debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:*
• No (disabled) if CONFIG_LWIP_DEBUG
**CONFIG_LWIP_DNS_DEBUG**

Enable DNS debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:* No (disabled) if CONFIG_LWIP_DEBUG

**CONFIG_LWIP_BRIDGEIF_DEBUG**

Enable bridge generic debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:* No (disabled) if CONFIG_LWIP_DEBUG

**CONFIG_LWIP_BRIDGEIF_FDB_DEBUG**

Enable bridge FDB debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:* No (disabled) if CONFIG_LWIP_DEBUG

**CONFIG_LWIP_BRIDGEIF_FW_DEBUG**

Enable bridge forwarding debug messages

*Found in:* Component config > LWIP > CONFIG_LWIP_DEBUG

*Default value:* No (disabled) if CONFIG_LWIP_DEBUG

### mbedTLS

Contains:

- `CONFIG_MBEDTLSASYMMETRICCONTENTLEN`
- Certificate Bundle
- Certificates
- `CONFIG_MBEDTLSCHACHA20_C`
- `CONFIG_MBEDTSLDHM_C`
- `CONFIG_MBEDTLSCECP_C`
- `CONFIG_MBEDTSECDFH_C`
- `CONFIG_MBEDTLSCECPAKE_C`
- `CONFIG_MBEDTLSCECPDPB256R1ENABLED`
- `CONFIG_MBEDTLSCECPDPB384R1ENABLED`
- `CONFIG_MBEDTLSCECPDPBP512R1ENABLED`
- `CONFIG_MBEDTLSCMAC_C`
- `CONFIG_MBEDTLSCECPDPCURVE25519ENABLED`
- `CONFIG_MBEDTSECDSADETERMINISTIC`
- `CONFIG_MBEDTLShardwareAES`
- `CONFIG_MBEDTLShardwareECC`
- `CONFIG_MBEDTLSATCA_HW_ECDSA_SIGN`
- `CONFIG_MBEDTLSATCA_HW_ECDSA_VERIFY`
- `CONFIG_MBEDTLSHardwareMPI`
- `CONFIG_MBEDTLSHardwareSHA`
- `CONFIG_MBEDTLSDEBUG`
- `CONFIG_MBEDTLSCECPRESTARTABLE`
- `CONFIG_MBEDTLSHAVE_TIME`
- `CONFIG_MBEDTSLRIPEMD160_C`
Chapter 2. API

- CONFIG_MBEDTLS_ECP_DP_SECP192K1_ENABLED
- CONFIG_MBEDTLS_ECP_DP_SECP192R1_ENABLED
- CONFIG_MBEDTLS_ECP_DP_SECP224K1_ENABLED
- CONFIG_MBEDTLS_ECP_DP_SECP224R1_ENABLED
- CONFIG_MBEDTLS_ECP_DP_SECP256K1_ENABLED
- CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED
- CONFIG_MBEDTLS_ECP_DP_SECP384R1_ENABLED
- CONFIG_MBEDTLS_ECP_DP_SECP521R1_ENABLED
- CONFIG_MBEDTLS_SHA512_C
- CONFIG_MBEDTLS_THREADING_C
- CONFIG_MBEDTLS_LARGE_KEYSOFTWARE_MPI
- CONFIG_MBEDTLS_HKDF_C
- mbedTLS v3.x related
- CONFIG_MBEDTLS_MEM_ALLOC_MODE
- CONFIG_MBEDTLS_ECP_NIST_OPTIM
- CONFIG_MBEDTLS_POLY1305_C
- CONFIG_MBEDTLS_SECURITY_RISKS
- CONFIG_MBEDTLS_SSL_ALPN
- CONFIG_MBEDTLS_SSL_PROTO_DTLS
- CONFIG_MBEDTLS_SSL_PROTO_GMTSSL1_1
- CONFIG_MBEDTLS_SSL_PROTO_TLS1_2
- CONFIG_MBEDTLS_SSL_RENEGOTIATION
- Symmetric Ciphers
- TLS Key Exchange Methods
- CONFIG_MBEDTLS_SSL_MAX_CONTENT_LEN
- CONFIG_MBEDTLS_TLS_MODE
- CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS
- CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS
- CONFIG_MBEDTLS_ROM_MD5
- CONFIG_MBEDTLS_DYNAMIC_BUFFER

**CONFIG_MBEDTLS_MEM_ALLOC_MODE**

Memory allocation strategy

*Found in: Component config > mbedTLS*

Allocation strategy for mbedTLS, essentially provides ability to allocate all required dynamic allocations from,

- Internal DRAM memory only
- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF
- Custom allocation mode, by overwriting calloc()/free() using mbedtls_platform_set_calloc_free() function
- Internal IRAM memory wherever applicable else internal DRAM

Recommended mode here is always internal ( *), since that is most preferred from security perspective. But if application requirement does not allow sufficient free internal memory then alternate mode can be selected.

(*) In case of ESP32-S2/ESP32-S3, hardware allows encryption of external SPIRAM contents provided hardware flash encryption feature is enabled. In that case, using external SPIRAM allocation strategy is also safe choice from security perspective.

**Available options:**

- Internal memory (MBEDTLS_INTERNAL_MEM_ALLOC)
- External SPIRAM (MBEDTLS_EXTERNAL_MEM_ALLOC)
- Default alloc mode (MBEDTLS_DEFAULT_MEM_ALLOC)
- Custom alloc mode (MBEDTLS_CUSTOM_MEM_ALLOC)
• Internal IRAM (MBEDTLS_IRAM_8BIT_MEM_ALLOC)
  Allows to use IRAM memory region as 8bit accessible region.
  TLS input and output buffers will be allocated in IRAM section which is 32bit aligned mem-
  ory. Every unaligned (8bit or 16bit) access will result in an exception and incur penalty of
  certain clock cycles per unaligned read/write.

**CONFIG_MBEDTLS_SSL_MAX_CONTENT_LEN**

TLS maximum message content length

*Found in: Component config > mbedTLS*

Maximum TLS message length (in bytes) supported by mbedTLS.

16384 is the default and this value is required to comply fully with TLS standards.

However you can set a lower value in order to save RAM. This is safe if the other end of the connection
supports Maximum Fragment Length Negotiation Extension (max_fragment_length, see RFC6066) or
you know for certain that it will never send a message longer than a certain number of bytes.

If the value is set too low, symptoms are a failed TLS handshake or a return value of
MBEDTLS_ERR_SSL_INVALID_RECORD (-0x7200).

*Range:*
  - from 512 to 16384

*Default value:*
  - 16384

**CONFIG_MBEDTLS_SSL_IN_CONTENT_LEN**

TLS maximum incoming fragment length

*Found in: Component config > mbedTLS > CONFIG_MBEDTLSASYMMETRIC_CONTENT_LEN*

This defines maximum incoming fragment length, overriding default maximum content length
(MBEDTLS_SSL_MAX_CONTENT_LEN).

*Range:*
  - from 512 to 16384

*Default value:*
  - 16384

**CONFIG_MBEDTLS_SSL_OUT_CONTENT_LEN**

TLS maximum outgoing fragment length

*Found in: Component config > mbedTLS > CONFIG_MBEDTLSASYMMETRIC_CONTENT_LEN*

This defines maximum outgoing fragment length, overriding default maximum content length
(MBEDTLS_SSL_MAX_CONTENT_LEN).

*Range:*
  - from 512 to 16384
Default value:
• 4096

CONFIG_MBEDTLS_DYNAMIC_BUFFER

Using dynamic TX/RX buffer

*Found in: Component config > mbedTLS*

Using dynamic TX/RX buffer. After enabling this option, mbedTLS will allocate TX buffer when need to send data and then free it if all data is sent, allocate RX buffer when need to receive data and then free it when all data is used or read by upper layer.

By default, when SSL is initialized, mbedTLS also allocate TX and RX buffer with the default value of “MBEDTLS_SSL_OUT_CONTENT_LEN” or “MBEDTLS_SSL_IN_CONTENT_LEN”, so to save more heap, users can set the options to be an appropriate value.

Default value:
• No (disabled) if CONFIG_MBEDTLS_SSL_PROTO_DTLS && CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH

CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA

Free private key and DHM data after its usage

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_DYNAMIC_BUFFER*

Free private key and DHM data after its usage in handshake process.

The option will decrease heap cost when handshake, but also lead to problem:

Because all certificate, private key and DHM data are freed so users should register certificate and private key to ssl config object again.

Default value:
• No (disabled) if CONFIG_MBEDTLS_DYNAMIC_BUFFER

CONFIG_MBEDTLS_DYNAMIC_FREE_CA_CERT

Free SSL CA certificate after its usage

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_DYNAMIC_BUFFER > CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA*

Free CA certificate after its usage in the handshake process. This option will decrease the heap footprint for the TLS handshake, but may lead to a problem: If the respective ssl object needs to perform the TLS handshake again, the CA certificate should once again be registered to the ssl object.

Default value:
• Yes (enabled) if CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA

CONFIG_MBEDTLS_DEBUG

Enable mbedTLS debugging

*Found in: Component config > mbedTLS*

Enable mbedTLS debugging functions at compile time.

If this option is enabled, you can include “mbedtls/esp_debug.h” and call mbedtls_esp_enable_debug_log() at runtime in order to enable mbedTLS debug output via the ESP log mechanism.

Default value:
• No (disabled)
**CONFIG_MBEDTLS_DEBUG_LEVEL**

Set mbedTLS debugging level

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_DEBUG*

Set mbedTLS debugging level

**Available options:**
- Warning (MBEDTLS_DEBUG_LEVEL_WARN)
- Info (MBEDTLS_DEBUG_LEVEL_INFO)
- Debug (MBEDTLS_DEBUG_LEVEL_DEBUG)
- Verbose (MBEDTLS_DEBUG_LEVEL_VERBOSE)

**mbedTLS v3.x related**  Contains:
- DTLS-based configurations
- CONFIG_MBEDTLS_SSL_CONTEXT_SERIALIZATION
- CONFIG_MBEDTLS_X509_TRUSTED_CERT_CALLBACK
- CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE
- CONFIG_MBEDTLS_SSL_PROTO_TLS1_3
- CONFIG_MBEDTLS_ECDH_LEGACY_CONTEXT
- CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH

**CONFIG_MBEDTLS_SSL_PROTO_TLS1_3**

Support TLS 1.3 protocol

*Found in: Component config > mbedTLS > mbedTLS v3.x related*

**Default value:**
- No (disabled) if CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE && CONFIG_MBEDTLS_DYNAMIC_BUFFER

**CONFIG_MBEDTLS_SSL_TLS1_3_COMPATIBILITY_MODE**

Enable TLS 1.3 middlebox compatibility mode

*Found in: Component config > mbedTLS > mbedTLS v3.x related > CONFIG_MBEDTLS_SSL_PROTO_TLS1_3*

**Default value:**
- Yes (enabled) if CONFIG_MBEDTLS_SSL_PROTO_TLS1_3

**CONFIG_MBEDTLS_SSL_VARIABLE_BUFFER_LENGTH**

Variable SSL buffer length

*Found in: Component config > mbedTLS > mbedTLS v3.x related*

This enables the SSL buffer to be resized automatically based on the negotiated maximum fragment length in each direction.

**Default value:**
- No (disabled)

**CONFIG_MBEDTLS_ECDH_LEGACY_CONTEXT**

Use a backward compatible ECDH context (Experimental)

*Found in: Component config > mbedTLS > mbedTLS v3.x related*

Use the legacy ECDH context format. Define this option only if you enable MBEDTLS_ECP_RESTARTABLE or if you want to access ECDH context fields directly.
Chapter 2. API

Default value:
- No (disabled) if `CONFIG_MBEDTLS_ECDH_C` && `CONFIG_MBEDTLS_ECP_RESTARTABLE`

**CONFIG_MBEDTLS_X509_TRUSTED_CERT_CALLBACK**

Enable trusted certificate callbacks

*Found in: Component config > mbedTLS > mbedTLS v3.x related*

Enables users to configure the set of trusted certificates through a callback instead of a linked list.

See mbedTLS documentation for required API and more details.

Default value:
- No (disabled)

**CONFIG_MBEDTLS_SSL_CONTEXT.Serialization**

Enable serialization of the TLS context structures

*Found in: Component config > mbedTLS > mbedTLS v3.x related*

Enable serialization of the TLS context structures. This is a local optimization in handling a single, potentially long-lived connection.

See mbedTLS documentation for required API and more details. Disabling this option will save some code size.

Default value:
- No (disabled)

**CONFIG_MBEDTLS_SSL_KEEP_PEER.CERTIFICATE**

Keep peer certificate after handshake completion

*Found in: Component config > mbedTLS > mbedTLS v3.x related*

Keep the peer's certificate after completion of the handshake. Disabling this option will save about 4kB of heap and some code size.

See mbedTLS documentation for required API and more details.

Default value:
- Yes (enabled) if `MBEDTLS_DYNAMIC_FREE_PEER_CERT`

**DTLS-based configurations**

Contains:
- `CONFIG_MBEDTLS_SSL_DTLS_SRTP`
- `CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID`

**CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID**

Support for the DTLS Connection ID extension

*Found in: Component config > mbedTLS > mbedTLS v3.x related > DTLS-based configurations*

Enable support for the DTLS Connection ID extension which allows to identify DTLS connections across changes in the underlying transport. The Connection ID extension is still in draft state. Refer: version draft-ietf-tls-dtls-connection-id-05

Default value:
- No (disabled) if `CONFIG_MBEDTLS_SSL_PROTO_DTLS`
**CONFIG_MBEDTLS_SSL_CID_IN_LEN_MAX**

Maximum length of CIDs used for incoming DTLS messages

*Found in:* Component config > mbedtls > mbedtls v3.x related > DTLS-based configurations > CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID

Maximum length of CIDs used for incoming DTLS messages

**Range:**
- from 0 to 32 if `CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID`

**Default value:**
- 32 if `CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID`

---

**CONFIG_MBEDTLS_SSL_CID_OUT_LEN_MAX**

Maximum length of CIDs used for outgoing DTLS messages

*Found in:* Component config > mbedtls > mbedtls v3.x related > DTLS-based configurations > CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID

Maximum length of CIDs used for outgoing DTLS messages

**Range:**
- from 0 to 32 if `CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID`

**Default value:**
- 32 if `CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID`

---

**CONFIG_MBEDTLS_SSL_CID_PADDING_GRANULARITY**

Record plaintext padding (for DTLS 1.2)

*Found in:* Component config > mbedtls > mbedtls v3.x related > DTLS-based configurations > CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID

Controls the use of record plaintext padding when using the Connection ID extension in DTLS 1.2.

The padding will always be chosen so that the length of the padded plaintext is a multiple of the value of this option.

**Notes:** A value of 1 means that no padding will be used for outgoing records. On systems lacking division instructions, a power of two should be preferred.

**Range:**
- from 0 to 32 if `CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID`

**Default value:**
- 16 if `CONFIG_MBEDTLS_SSL_DTLS_CONNECTION_ID`

---

**CONFIG_MBEDTLS_SSL_DTLS_SRTP**

Enable support for negotiation of DTLS-SRTP (RFC 5764)

*Found in:* Component config > mbedtls > mbedtls v3.x related > DTLS-based configurations

Enable support for negotiation of DTLS-SRTP (RFC 5764) through the use _srtp extension.

See mbedtls documentation for required API and more details. Disabling this option will save some code size.

**Default value:**
- No (disabled) if `CONFIG_MBEDTLS_SSL_PROTO_DTLS`

---

**Certificate Bundle** Contains:

- `CONFIG_MBEDTLS_CERTIFICATE_BUNDLE`
Chapter 2. API

**CONFIG_MBEDTLS_CERTIFICATE_BUNDLE**

Enable trusted root certificate bundle

*Found in: Component config > mbedTLS > Certificate Bundle*

Enable support for large number of default root certificates

When enabled this option allows user to store default as well as customer specific root certificates in compressed format rather than storing full certificate. For the root certificates the public key and the subject name will be stored.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_DEFAULT_CERTIFICATE_BUNDLE**

Default certificate bundle options

*Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*

**Available options:**
- Use the full default certificate bundle (MBEDTLS_CERTIFICATE_BUNDLE_DEFAULT_FULL)
- Use only the most common certificates from the default bundles (MBEDTLS_CERTIFICATE_BUNDLE_DEFAULT_CMN)
  
  Use only the most common certificates from the default bundles, reducing the size with 50%, while still having around 99% coverage.
- Do not use the default certificate bundle (MBEDTLS_CERTIFICATE_BUNDLE_DEFAULT_NONE)

**CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE**

Add custom certificates to the default bundle

*Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*

**Default value:**
- No (disabled)

**CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE_PATH**

Custom certificate bundle path

*Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE > CONFIG_MBEDTLS_CUSTOM_CERTIFICATE_BUNDLE*

Name of the custom certificate directory or file. This path is evaluated relative to the project root directory.

**CONFIG_MBEDTLS_CERTIFICATE_BUNDLE_MAX_CERTS**

Maximum no of certificates allowed in certificate bundle

*Found in: Component config > mbedTLS > Certificate Bundle > CONFIG_MBEDTLS_CERTIFICATE_BUNDLE*

**Default value:**
- 200
CONFIG_MBEDTLS_ECP_RESTARTABLE
Enable mbedtls ecp restartable

*Found in: Component config > mbedtls*
Enable “non-blocking” ECC operations that can return early and be resumed.

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_CMAC_C
Enable CMAC mode for block ciphers

*Found in: Component config > mbedtls*
Enable the CMAC (Cipher-based Message Authentication Code) mode for block ciphers.

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_HARDWARE_AES
Enable hardware AES acceleration

*Found in: Component config > mbedtls*
Enable hardware accelerated AES encryption & decryption. Note that if the ESP32 CPU is running at 240MHz, hardware AES does not offer any speed boost over software AES.

**Default value:**
- Yes (enabled) if SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST

CONFIG_MBEDTLS_HARDWARE_GCM
Enable partially hardware accelerated GCM

*Found in: Component config > mbedtls > CONFIG_MBEDTLS_HARDWARE_AES*
Enable partially hardware accelerated GCM. GHASH calculation is still done in software. If MBEDTLS_HARDWARE_GCM is disabled and MBEDTLS_HARDWARE_AES is enabled then mbedtls will still use the hardware accelerated AES block operation, but on a single block at a time.

**Default value:**
- Yes (enabled) if SOC_AES_SUPPORT_GCM && CONFIG_MBEDTLS_HARDWARE_AES

CONFIG_MBEDTLS_HARDWARE_MPI
Enable hardware MPI (bignum) acceleration

*Found in: Component config > mbedtls*
Enable hardware accelerated multiple precision integer operations. Hardware accelerated multiplication, modulo multiplication, and modular exponentiation for up to SOC_RSA_MAX_BIT_LEN bit results. These operations are used by RSA.

**Default value:**
- Yes (enabled) if SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST
**CONFIG_MBEDTLS_HARDWARE_SHA**

Enable hardware SHA acceleration

*Found in: Component config > mbedTLS*

Enable hardware accelerated SHA1, SHA256, SHA384 & SHA512 in mbedTLS.

Due to a hardware limitation, on the ESP32 hardware acceleration is only guaranteed if SHA digests are calculated one at a time. If more than one SHA digest is calculated at the same time, one will be calculated fully in hardware and the rest will be calculated (at least partially calculated) in software. This happens automatically.

SHA hardware acceleration is faster than software in some situations but slower in others. You should benchmark to find the best setting for you.

**Default value:**
- Yes (enabled) if SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST

**CONFIG_MBEDTLS_HARDWARE_ECC**

Enable hardware ECC acceleration

*Found in: Component config > mbedTLS*

Enable hardware accelerated ECC point multiplication and point verification for points on curve SECP192R1 and SECP256R1 in mbedTLS

**Default value:**
- Yes (enabled) if SOC_ECC_SUPPORTED

**CONFIG_MBEDTLS_ECC_OTHER_CURVES_SOFT_FALLBACK**

Fallback to software implementation for curves not supported in hardware

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HARDWARE_ECC*

Fallback to software implementation of ECC point multiplication and point verification for curves not supported in hardware.

**Default value:**
- Yes (enabled) if CONFIG_MBEDTLS_HARDWARE_ECC

**CONFIG_MBEDTLS_ROM_MD5**

Use MD5 implementation in ROM

*Found in: Component config > mbedTLS*

Use ROM MD5 in mbedTLS.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN**

Enable hardware ECDSA sign acceleration when using ATECC608A

*Found in: Component config > mbedTLS*

This option enables hardware acceleration for ECDSA sign function, only when using ATECC608A cryptoauth chip (integrated with ESP32-WROOM-32SE)

**Default value:**
- No (disabled)
CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY
Enable hardware ECDSA verify acceleration when using ATECC608A

*Found in: Component config > mbedTLS*

This option enables hardware acceleration for ECDSA sign function, only when using ATECC608A cryptoauth chip (integrated with ESP32-WROOM-32SE)

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_HAVE_TIME
Enable mbedtls time support

*Found in: Component config > mbedTLS*

Enable use of time.h functions (time() and gmtime()) by mbedTLS.

This option doesn’t require the system time to be correct, but enables functionality that requires relative timekeeping - for example periodic expiry of TLS session tickets or session cache entries.

Disabling this option will save some firmware size, particularly if the rest of the firmware doesn’t call any standard timekeeping functions.

**Default value:**
- Yes (enabled)

CONFIG_MBEDTLS_PLATFORM_TIME_ALT
Enable mbedtls time support: platform-specific

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HAVE_TIME*

Enabling this config will provide users with a function `mbedtls_platform_set_time()` that allows to set an alternative time function pointer.

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_HAVE_TIME_DATE
Enable mbedtls certificate expiry check

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HAVE_TIME*

Enables X.509 certificate expiry checks in mbedTLS.

If this option is disabled (default) then X.509 certificate “valid from” and “valid to” timestamp fields are ignored.

If this option is enabled, these fields are compared with the current system date and time. The time is retrieved using the standard time() and gmtime() functions. If the certificate is not valid for the current system time then verification will fail with code MBEDTLS_X509_BADCERT_FUTURE or MBEDTLS_X509_BADCERT_EXPIRED.

Enabling this option requires adding functionality in the firmware to set the system clock to a valid timestamp before using TLS. The recommended way to do this is via ESP-IDF’s SNTP functionality, but any method can be used.

In the case where only a small number of certificates are trusted by the device, please carefully consider the tradeoffs of enabling this option. There may be undesired consequences, for example if all trusted certificates expire while the device is offline and a TLS connection is required to update. Or if an issue with the SNTP server means that the system time is invalid for an extended period after a reset.

**Default value:**
Chapter 2. API 参考

- No (disabled)

**CONFIG_MBEDTLS_ECDSA_DETERMINISTIC**

Enable deterministic ECDSA

*Found in: Component config > mbedTLS*

Standard ECDSA is “fragile” in the sense that lack of entropy when signing may result in a compromise of the long-term signing key.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_SHA512_C**

Enable the SHA-384 and SHA-512 cryptographic hash algorithms

*Found in: Component config > mbedTLS*

Enable MBEDTLS_SHA512_C adds support for SHA-384 and SHA-512.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_TLS_MODE**

TLS Protocol Role

*Found in: Component config > mbedTLS*

mbedTLS can be compiled with protocol support for the TLS server, TLS client, or both server and client.

Reducing the number of TLS roles supported saves code size.

**Available options:**
- Server & Client (MBEDTLS_TLS_SERVER_AND_CLIENT)
- Server (MBEDTLS_TLS_SERVER_ONLY)
- Client (MBEDTLS_TLS_CLIENT_ONLY)
- None (MBEDTLS_TLS_DISABLED)

**TLS Key Exchange Methods**  Contains:

- `CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_RSA`
- `CONFIG_MBEDTLS_KEY_EXCHANGE_ECJPAKE`
- `CONFIG_MBEDTLS_PSK_MODES`
- `CONFIG_MBEDTLS_KEY_EXCHANGE_RSA`
- `CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE`

**CONFIG_MBEDTLS_PSK_MODES**

Enable pre-shared-key ciphersuites

*Found in: Component config > mbedTLS > TLS Key Exchange Methods*

Enable to show configuration for different types of pre-shared-key TLS authentication methods.

Leaving this options disabled will save code size if they are not used.

**Default value:**
- No (disabled)
CONFIG_MBEDTLS_KEY_EXCHANGE_PSK

Enable PSK based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES*

Enable to support symmetric key PSK (pre-shared-key) TLS key exchange modes.

*Default value:*
  - No (disabled) if CONFIG_MBEDTLS_PSK_MODES

CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_PSK

Enable DHE-PSK based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES*

Enable to support Diffie-Hellman PSK (pre-shared-key) TLS authentication modes.

*Default value:*
  - Yes (enabled) if CONFIG_MBEDTLS_PSK_MODES && CONFIG_MBEDTLS_DHM_C

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_PSK

Enable ECDHE-PSK based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES*

Enable to support Elliptic-Curve-Diffie-Hellman PSK (pre-shared-key) TLS authentication modes.

*Default value:*
  - Yes (enabled) if CONFIG_MBEDTLS_PSK_MODES && CONFIG_MBEDTLS_ECDH_C

CONFIG_MBEDTLS_KEY_EXCHANGE_RSA_PSK

Enable RSA-PSK based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES*

Enable to support RSA PSK (pre-shared-key) TLS authentication modes.

*Default value:*
  - Yes (enabled) if CONFIG_MBEDTLS_PSK_MODES

CONFIG_MBEDTLS_KEY_EXCHANGE_RSA

Enable RSA-only based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods*

Enable to support ciphersuites with prefix TLS-RSA-WITH-

*Default value:*
  - Yes (enabled)

CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_RSA

Enable DHE-RSA based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods*

Enable to support ciphersuites with prefix TLS-DHE-RSA-WITH-
Default value:
  • Yes (enabled) if `CONFIG_MBEDTLS_DHМ_C`

**CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE**

Support Elliptic Curve based ciphersuites

*Found in: Component config > mbedTLS > TLS Key Exchange Methods*

Enable to show Elliptic Curve based ciphersuite mode options.

Disabling all Elliptic Curve ciphersuites saves code size and can give slightly faster TLS handshakes, provided the server supports RSA-only ciphersuite modes.

**Default value:**
  • Yes (enabled)

**CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_RSA**

Enable ECDHE-RSA based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE*

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

**Default value:**
  • Yes (enabled)

**CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_ECDSA**

Enable ECDHE-ECDSA based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE*

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

**Default value:**
  • Yes (enabled)

**CONFIG_MBEDTLS_KEY_EXCHANGE_ECDH_ECDSA**

Enable ECDH-ECDSA based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE*

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

**Default value:**
  • Yes (enabled)

**CONFIG_MBEDTLS_KEY_EXCHANGE_ECDH_RSA**

Enable ECDH-RSA based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE*

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

**Default value:**
  • Yes (enabled)
CONFIG_MBEDTLS_KEY_EXCHANGE_ECJPAKE

Enable ECJPAKE based ciphersuite modes

*Found in: Component config > mbedTLS > TLS Key Exchange Methods*

Enable to support ciphersuites with prefix TLS-ECJPAKE-WITH-

**Default value:**
- No (disabled) if `CONFIG_MBEDTLS_ECJPAKE_C` && `CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED`

CONFIG_MBEDTLS_SSL_RENEGOTIATION

Support TLS renegotiation

*Found in: Component config > mbedTLS*

The two main uses of renegotiation are (1) refresh keys on long-lived connections and (2) client authentication after the initial handshake. If you don’t need renegotiation, disabling it will save code size and reduce the possibility of abuse/vulnerability.

**Default value:**
- Yes (enabled)

CONFIG_MBEDTLS_SSL_PROTO_TLS1_2

Support TLS 1.2 protocol

*Found in: Component config > mbedTLS*

**Default value:**
- Yes (enabled)

CONFIG_MBEDTLS_SSL_PROTO_GMTSSL1_1

Support GM/T SSL 1.1 protocol

*Found in: Component config > mbedTLS*

Provisions for GM/T SSL 1.1 support

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_SSL_PROTO DTLS

Support DTLS protocol (all versions)

*Found in: Component config > mbedTLS*

Requires TLS 1.2 to be enabled for DTLS 1.2

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_SSL_ALPN

Support ALPN (Application Layer Protocol Negotiation)

*Found in: Component config > mbedTLS*

Disabling this option will save some code size if it is not needed.

**Default value:**
- Yes (enabled)
**CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS**

TLS: Client Support for RFC 5077 SSL session tickets

*Found in: Component config > mbedTLS*

Client support for RFC 5077 session tickets. See mbedTLS documentation for more details. Disabling this option will save some code size.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS**

TLS: Server Support for RFC 5077 SSL session tickets

*Found in: Component config > mbedTLS*

Server support for RFC 5077 session tickets. See mbedTLS documentation for more details. Disabling this option will save some code size.

**Default value:**
- Yes (enabled)

**Symmetric Ciphers**

Contains:
- `CONFIG_MBEDTLS_AES_C`
- `CONFIG_MBEDTLS_BLOWFISH_C`
- `CONFIG_MBEDTLS_CAMELLIA_C`
- `CONFIG_MBEDTLS_CCM_C`
- `CONFIG_MBEDTLS_DES_C`
- `CONFIG_MBEDTLS_GCM_C`
- `CONFIG_MBEDTLS_NIST_KW_C`
- `CONFIG_MBEDTLS_XTEA_C`

**CONFIG_MBEDTLS_AES_C**

AES block cipher

*Found in: Component config > mbedTLS > Symmetric Ciphers*

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_CAMELLIA_C**

Camellia block cipher

*Found in: Component config > mbedTLS > Symmetric Ciphers*

**Default value:**
- No (disabled)

**CONFIG_MBEDTLS_DES_C**

DES block cipher (legacy, insecure)

*Found in: Component config > mbedTLS > Symmetric Ciphers*

Enables the DES block cipher to support 3DES-based TLS ciphersuites. 3DES is vulnerable to the Sweet32 attack and should only be enabled if absolutely necessary.

**Default value:**
- No (disabled)
CONFIG_MBEDTLS_BLOWFISH_C
Blowfish block cipher (read help)

*Found in: Component config > mbedTLS > Symmetric Ciphers*

Enables the Blowfish block cipher (not used for TLS sessions.)
The Blowfish cipher is not used for mbedTLS TLS sessions but can be used for other purposes. Read up on the limitations of Blowfish (including Sweet32) before enabling.

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_XTEA_C
XTEA block cipher

*Found in: Component config > mbedTLS > Symmetric Ciphers*

Enables the XTEA block cipher.

**Default value:**
- No (disabled)

CONFIG_MBEDTLS_CCM_C
CCM (Counter with CBC-MAC) block cipher modes

*Found in: Component config > mbedTLS > Symmetric Ciphers*

Enable Counter with CBC-MAC (CCM) modes for AES and/or Camellia ciphers.

Disabling this option saves some code size.

**Default value:**
- Yes (enabled)

CONFIG_MBEDTLS_GCM_C
GCM (Galois/Counter) block cipher modes

*Found in: Component config > mbedTLS > Symmetric Ciphers*

Enable Galois/Counter Mode for AES and/or Camellia ciphers.

This option is generally faster than CCM.

**Default value:**
- Yes (enabled)

CONFIG_MBEDTLS_NIST_KW_C
NIST key wrapping (KW) and KW padding (KWP)

*Found in: Component config > mbedTLS > Symmetric Ciphers*

Enable NIST key wrapping and key wrapping padding.

**Default value:**
- No (disabled)
**CONFIG_MBEDTLS_RIPEMD160_C**

Enable RIPEMD-160 hash algorithm

*Found in: Component config > mbedTLS*

Enable the RIPEMD-160 hash algorithm.

**Default value:**
- No (disabled)

**Certificates**

Contains:
- **CONFIG_MBEDTLS_PEM_PARSE_C**
- **CONFIG_MBEDTLS_PEM_WRITE_C**
- **CONFIG_MBEDTLS_X509_CRL_PARSE_C**
- **CONFIG_MBEDTLS_X509_CSRPARSE_C**

**CONFIG_MBEDTLS_PEM_PARSE_C**

Read & Parse PEM formatted certificates

*Found in: Component config > mbedTLS > Certificates*

Enable decoding/parsing of PEM formatted certificates.

If your certificates are all in the simpler DER format, disabling this option will save some code size.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_PEM_WRITE_C**

Write PEM formatted certificates

*Found in: Component config > mbedTLS > Certificates*

Enable writing of PEM formatted certificates.

If writing certificate data only in DER format, disabling this option will save some code size.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_X509_CRL_PARSE_C**

X.509 CRL parsing

*Found in: Component config > mbedTLS > Certificates*


**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_X509_CSRPARSE_C**

X.509 CSR parsing

*Found in: Component config > mbedTLS > Certificates*

Support for parsing X.509 Certificate Signing Requests

**Default value:**
- Yes (enabled)
Chapter 2. API Reference

**CONFIG_MBEDTLS_ECP_C**
Elliptic Curve Ciphers
*Found in: Component config > mbedTLS*

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_DH_C**
Diffie-Hellman-Merkle key exchange (DHM)
*Found in: Component config > mbedTLS*

Enable DHM. Needed to use DHE-xxx TLS ciphersuites.

Note that the security of Diffie-Hellman key exchanges depends on a suitable prime being used for the exchange. Please see detailed warning text about this in file `mbedtls/dhm.h` file.

**Default value:**
- No (disabled)

**CONFIG_MBEDTLS_ECDH_C**
Elliptic Curve Diffie-Hellman (ECDH)
*Found in: Component config > mbedTLS*

Enable ECDH. Needed to use ECDHE-xxx TLS ciphersuites.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_ECDSA_C**
Elliptic Curve DSA
*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECDH_C*

Enable ECDSA. Needed to use ECDSA-xxx TLS ciphersuites.

**Default value:**
- Yes (enabled)

**CONFIG_MBEDTLS_ECJPAKE_C**
Elliptic curve J-PAKE
*Found in: Component config > mbedTLS*

Enable ECJPAKE. Needed to use ECJPAKE-xxx TLS ciphersuites.

**Default value:**
- No (disabled)

**CONFIG_MBEDTLS_ECP_DP_SECP192R1_ENABLED**
Enable SECP192R1 curve
*Found in: Component config > mbedTLS*

Enable support for SECP192R1 Elliptic Curve.

**Default value:**
- Yes (enabled) if `CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN || CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY` & `CONFIG_MBEDTLS_ECP_C`
CONFIG_MBEDTLS_ECP_DP_SECP224R1_ENABLED
Enable SECP224R1 curve

Default value:
- Yes (enabled) if \((CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_SIGN \mid CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_VERIFY) \land CONFIG\_MBEDTLS\_ECP\_C\)

CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED
Enable SECP256R1 curve

Default value:
- Yes (enabled)

CONFIG_MBEDTLS_ECP_DP_SECP384R1_ENABLED
Enable SECP384R1 curve

Default value:
- Yes (enabled) if \((CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_SIGN \mid CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_VERIFY) \land CONFIG\_MBEDTLS\_ECP\_C\)

CONFIG_MBEDTLS_ECP_DP_SECP521R1_ENABLED
Enable SECP521R1 curve

Default value:
- Yes (enabled) if \((CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_SIGN \mid CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_VERIFY) \land CONFIG\_MBEDTLS\_ECP\_C\)

CONFIG_MBEDTLS_ECP_DP_SECP192K1_ENABLED
Enable SECP192K1 curve

Default value:
- Yes (enabled) if \((CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_SIGN \mid CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_VERIFY) \land CONFIG\_MBEDTLS\_ECP\_C\)

CONFIG_MBEDTLS_ECP_DP_SECP224K1_ENABLED
Enable SECP224K1 curve

Default value:
- Yes (enabled) if \((CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_SIGN \mid CONFIG\_MBEDTLS\_ATCA\_HW\_ECDSA\_VERIFY) \land CONFIG\_MBEDTLS\_ECP\_C\)
**Default value:**
- Yes (enabled) if `(CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN || CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY) && CONFIG_MBEDTLS_ECP_C`

**CONFIG_MBEDTLS_ECP_DP_SECP256K1_ENABLED**

Enable SECP256K1 curve

*Found in: Component config > mbedTLS*

Enable support for SECP256K1 Elliptic Curve.

**Default value:**
- Yes (enabled) if `(CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN || CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY) && CONFIG_MBEDTLS_ECP_C`

**CONFIG_MBEDTLS_ECP_DP_BP256R1_ENABLED**

Enable BP256R1 curve

*Found in: Component config > mbedTLS*

support for DP Elliptic Curve.

**Default value:**
- Yes (enabled) if `(CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN || CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY) && CONFIG_MBEDTLS_ECP_C`

**CONFIG_MBEDTLS_ECP_DP_BP384R1_ENABLED**

Enable BP384R1 curve

*Found in: Component config > mbedTLS*

support for DP Elliptic Curve.

**Default value:**
- Yes (enabled) if `(CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN || CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY) && CONFIG_MBEDTLS_ECP_C`

**CONFIG_MBEDTLS_ECP_DP_BP512R1_ENABLED**

Enable BP512R1 curve

*Found in: Component config > mbedTLS*

support for DP Elliptic Curve.

**Default value:**
- Yes (enabled) if `(CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN || CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY) && CONFIG_MBEDTLS_ECP_C`

**CONFIG_MBEDTLS_ECP_DP_CURVE25519_ENABLED**

Enable CURVE25519 curve

*Found in: Component config > mbedTLS*

Enable support for CURVE25519 Elliptic Curve.

**Default value:**
- Yes (enabled) if `(CONFIG_MBEDTLS_ATCA_HW_ECDSA_SIGN || CONFIG_MBEDTLS_ATCA_HW_ECDSA_VERIFY) && CONFIG_MBEDTLS_ECP_C`
CONFIG_MBEDTLS_ECP_NIST_OPTIM

NIST ‘modulo p’ optimisations

*Found in: Component config > mbedTLS*

NIST ‘modulo p’ optimisations increase Elliptic Curve operation performance.
Disabling this option saves some code size.

*Default value:*
  - Yes (enabled)

CONFIG_MBEDTLS_POLY1305_C

Poly1305 MAC algorithm

*Found in: Component config > mbedTLS*

Enable support for Poly1305 MAC algorithm.

*Default value:*
  - No (disabled)

CONFIG_MBEDTLS_CHACHA20_C

Chacha20 stream cipher

*Found in: Component config > mbedTLS*

Enable support for Chacha20 stream cipher.

*Default value:*
  - No (disabled)

CONFIG_MBEDTLS_CHACHA保利_C

ChaCha20-Poly1305 AEAD algorithm

*Found in: Component config > mbedTLS>* CONFIG_MBEDTLS_CHACHA20_C

Enable support for ChaCha20-Poly1305 AEAD algorithm.

*Default value:*
  - No (disabled) if CONFIG_MBEDTLS_CHACHA20_C && CONFIG_MBEDTLS_POLY1305_C

CONFIG_MBEDTLS_HKDF_C

HKDF algorithm (RFC 5869)

*Found in: Component config > mbedTLS*

Enable support for the Hashed Message Authentication Code (HMAC)-based key derivation function (HKDF).

*Default value:*
  - No (disabled)
CONFIG_MBEDTLS_THREADING_C
Enable the threading abstraction layer

*Found in: Component config > mbedTLS*

If you do intend to use contexts between threads, you will need to enable this layer to prevent race conditions.

*Default value:*
  * No (disabled)*

CONFIG_MBEDTLS_THREADING_ALT
Enable threading alternate implementation

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_THREADING_C*

Enable threading alt to allow your own alternate threading implementation.

*Default value:*
  * Yes (enabled) if CONFIG_MBEDTLS_THREADING_C*

CONFIG_MBEDTLS_THREADING_PTHREAD
Enable threading pthread implementation

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_THREADING_C*

Enable the pthread wrapper layer for the threading layer.

*Default value:*
  * No (disabled) if CONFIG_MBEDTLS_THREADING_C*

CONFIG_MBEDTLS_LARGE_KEY_SOFTWARE_MPI
Fallback to software implementation for larger MPI values

*Found in: Component config > mbedTLS*

Fallback to software implementation for RSA key lengths larger than SOC_RSA_MAX_BIT_LEN. If this is not active then the ESP will be unable to process keys greater than SOC_RSA_MAX_BIT_LEN.

*Default value:*
  * No (disabled)*

CONFIG_MBEDTLS_SECURITY_RISKS
Show configurations with potential security risks

*Found in: Component config > mbedTLS*

*Default value:*
  * No (disabled)*

Contains:
  * CONFIG_MBEDTLS_ALLOW_UNSUPPORTED_CRITICAL_EXT*

CONFIG_MBEDTLS_ALLOW_UNSUPPORTED_CRITICAL_EXT
X.509 CRT parsing with unsupported critical extensions

*Found in: Component config > mbedTLS > CONFIG_MBEDTLS_SECURITY_RISKS*

Allow the X.509 certificate parser to load certificates with unsupported critical extensions
Default value:
• No (disabled) if `CONFIG_MBEDTLS_SECURITY_RISKS`

ESP-MQTT Configurations Contains:
• `CONFIG_MQTT_CUSTOM_OUTBOX`
• `CONFIG_MQTT_TRANSPORT_SSL`
• `CONFIG_MQTT_TRANSPORT_WEBSOCKET`
• `CONFIG_MQTT_PROTOCOL_311`
• `CONFIG_MQTT_PROTOCOL_5`
• `CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED`
• `CONFIG_MQTT_USE_CUSTOM_CONFIG`
• `CONFIG_MQTT_OUTBOX_EXPIRED_TIMEOUT_MS`
• `CONFIG_MQTT_REPORT_DELETED_MESSAGES`
• `CONFIG_MQTT_SKIP_PUBLISH_IF_DISCONNECTED`
• `CONFIG_MQTT_MSG_ID_INCREMENTAL`

**CONFIG_MQTT_PROTOCOL_311**
Enable MQTT protocol 3.1.1

*Found in: Component config > ESP-MQTT Configurations*

If not, this library will use MQTT protocol 3.1

Default value:
• Yes (enabled)

**CONFIG_MQTT_PROTOCOL_5**
Enable MQTT protocol 5.0

*Found in: Component config > ESP-MQTT Configurations*

If not, this library will not support MQTT 5.0

Default value:
• No (disabled)

**CONFIG_MQTT_TRANSPORT_SSL**
Enable MQTT over SSL

*Found in: Component config > ESP-MQTT Configurations*

Enable MQTT transport over SSL with mbedTLS

Default value:
• Yes (enabled)

**CONFIG_MQTT_TRANSPORT_WEBSOCKET**
Enable MQTT over Websocket

*Found in: Component config > ESP-MQTT Configurations*

Enable MQTT transport over Websocket.

Default value:
• Yes (enabled)
CONFIG_MQTT_TRANSPORT_WEBSOCKET_SECURE

Enable MQTT over Websocket Secure

Found in: Component config > ESP-MQTT Configurations

Enable MQTT transport over Websocket Secure.

Default value:
- Yes (enabled)

CONFIG_MQTT_MSG_ID_INCREMENTAL

Use Incremental Message Id

Found in: Component config > ESP-MQTT Configurations

Set this to true for the message id (2.3.1 Packet Identifier) to be generated as an incremental number rather than a random value (used by default)

Default value:
- No (disabled)

CONFIG_MQTT_SKIP_PUBLISH_IF_DISCONNECTED

Skip publish if disconnected

Found in: Component config > ESP-MQTT Configurations

Set this to true to avoid publishing (enqueueing messages) if the client is disconnected. The MQTT client tries to publish all messages by default, even in the disconnected state (where the qos1 and qos2 packets are stored in the internal outbox to be published later) The MQTT_SKIP_PUBLISH_IF_DISCONNECTED option allows applications to override this behaviour and not enqueue publish packets in the disconnected state.

Default value:
- No (disabled)

CONFIG_MQTT_REPORT_DELETED_MESSAGES

Report deleted messages

Found in: Component config > ESP-MQTT Configurations

Set this to true to post events for all messages which were deleted from the outbox before being correctly sent and confirmed.

Default value:
- No (disabled)

CONFIG_MQTT_USE_CUSTOM_CONFIG

MQTT Using custom configurations

Found in: Component config > ESP-MQTT Configurations

Custom MQTT configurations.

Default value:
- No (disabled)
**CONFIG_MQTT_TCP_DEFAULT_PORT**

Default MQTT over TCP port

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

Default MQTT over TCP port

*Default value:*

- 1883 if `CONFIG_MQTT_USE_CUSTOM_CONFIG`

**CONFIG_MQTT_SSL_DEFAULT_PORT**

Default MQTT over SSL port

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

Default MQTT over SSL port

*Default value:*

- 8883 if `CONFIG_MQTT_USE_CUSTOM_CONFIG` & & `CONFIG_MQTT_TRANSPORT_SSL`

**CONFIG_MQTT_WS_DEFAULT_PORT**

Default MQTT over Websocket port

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

Default MQTT over Websocket port

*Default value:*

- 80 if `CONFIG_MQTT_USE_CUSTOM_CONFIG` & & `CONFIG_MQTT_TRANSPORT_WEBSOCKET`

**CONFIG_MQTT_WS_DEFAULT_PORT**

Default MQTT over Websocket Secure port

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

Default MQTT over Websocket Secure port

*Default value:*

- 443 if `CONFIG_MQTT_USE_CUSTOM_CONFIG` & & `CONFIG_MQTT_TRANSPORT_WEBSOCKET` & & `CONFIG_MQTT_TRANSPORT_WEBSOCKET_SECURE`

**CONFIG_MQTT_BUFFER_SIZE**

Default MQTT Buffer Size

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

This buffer size using for both transmit and receive

*Default value:*

- 1024 if `CONFIG_MQTT_USE_CUSTOM_CONFIG`

**CONFIG_MQTT_TASK_STACK_SIZE**

MQTT task stack size

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

MQTT task stack size

*Default value:*

- 6144 if `CONFIG_MQTT_USE_CUSTOM_CONFIG`
Chapter 2. API

**CONFIG_MQTT_DISABLE_API_LOCKS**

Disable API locks

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

Default config employs API locks to protect internal structures. It is possible to disable these locks if the user code doesn’t access MQTT API from multiple concurrent tasks

**Default value:**
- No (disabled) if `CONFIG_MQTT_USE_CUSTOM_CONFIG`

**CONFIG_MQTT_TASK_PRIORITY**

MQTT task priority

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG*

MQTT task priority. Higher number denotes higher priority.

**Default value:**
- 5 if `CONFIG_MQTT_USE_CUSTOM_CONFIG`

**CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED**

Enable MQTT task core selection

*Found in: Component config > ESP-MQTT Configurations*

This will enable core selection

**CONFIG_MQTT_TASK_CORE_SELECTION**

Core to use?

*Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED*

**Available options:**
- Core 0 (MQTT_USE_CORE_0)
- Core 1 (MQTT_USE_CORE_1)

**CONFIG_MQTT_CUSTOM_OUTBOX**

Enable custom outbox implementation

*Found in: Component config > ESP-MQTT Configurations*

Set to true if a specific implementation of message outbox is needed (e.g. persistent outbox in NVM or similar). Note: Implementation of the custom outbox must be added to the mqtt component. These CMake commands could be used to append the custom implementation to lib-mqtt sources:

```
idf_component_get_property(mqtt)
set_property(TARGET ${mqtt}
PROPERTY SOURCES ${PROJECT_DIR}/custom_outbox.c APPEND)
```

**Default value:**
- No (disabled)

**CONFIG_MQTT_OUTBOX_EXPIRED_TIMEOUT_MS**

Outbox message expired timeout[ms]

*Found in: Component config > ESP-MQTT Configurations*

Messages which stays in the outbox longer than this value before being published will be discarded.

**Default value:**
Chapter 2. API

• 30000 if \texttt{CONFIG_MQTT_USE_CUSTOM_CONFIG}

\textbf{Newlib} Contains:

• \texttt{CONFIG_NEWLIB_NANO_FORMAT}
• \texttt{CONFIG_NEWLIB_STDIN_LINE_ENDING}
• \texttt{CONFIG_NEWLIB_STDOUT_LINE_ENDING}
• \texttt{CONFIG_NEWLIB_TIME_SYSCALL}

\textbf{CONFIG_NEWLIB_STDOUT_LINE_ENDING}

Line ending for UART output

\textit{Found in: Component config > Newlib}

This option allows configuring the desired line endings sent to UART when a newline (‘\n’, LF) appears on stdout. Three options are possible:

CRLF: whenever LF is encountered, prepend it with CR
LF: no modification is applied, stdout is sent as is
CR: each occurrence of LF is replaced with CR

This option doesn’t affect behavior of the UART driver (drivers/uart.h).

\textbf{Available options:}

• CRLF (NEWLIB_STDOUT_LINE_ENDING_CRLF)
• LF (NEWLIB_STDOUT_LINE_ENDING_LF)
• CR (NEWLIB_STDOUT_LINE_ENDING_CR)

\textbf{CONFIG_NEWLIB_STDIN_LINE_ENDING}

Line ending for UART input

\textit{Found in: Component config > Newlib}

This option allows configuring which input sequence on UART produces a newline (‘\n’, LF) on stdin. Three options are possible:

CRLF: CRLF is converted to LF
LF: no modification is applied, input is sent to stdin as is
CR: each occurrence of CR is replaced with LF

This option doesn’t affect behavior of the UART driver (drivers/uart.h).

\textbf{Available options:}

• CRLF (NEWLIB_STDIN_LINE_ENDING_CRLF)
• LF (NEWLIB_STDIN_LINE_ENDING_LF)
• CR (NEWLIB_STDIN_LINE_ENDING_CR)

\textbf{CONFIG_NEWLIB_NANO_FORMAT}

Enable ‘nano’ formatting options for printf/scanf family

\textit{Found in: Component config > Newlib}

ESP32 ROM contains parts of newlib C library, including printf/scanf family of functions. These functions have been compiled with so-called “nano” formatting option. This option doesn’t support 64-bit integer formats and C99 features, such as positional arguments.

For more details about “nano” formatting option, please see newlib readme file, search for ‘–enable-newlib-nano-formatted-io’ : \url{https://sourceware.org/newlib/README}
If this option is enabled, build system will use functions available in ROM, reducing the application binary size. Functions available in ROM run faster than functions which run from flash. Functions available in ROM can also run when flash instruction cache is disabled.

If you need 64-bit integer formatting support or C99 features, keep this option disabled.

**CONFIG_NEWLIB_TIME_SYSCALL**

Timers used for gettimeofday function

*Found in: Component config > Newlib*

This setting defines which hardware timers are used to implement ‘gettimeofday’ and ‘time’ functions in C library.

- **If both high-resolution (systimer for all targets except ESP32) and RTC timers are used**, timekeeping will continue in deep sleep. Time will be reported at 1 microsecond resolution. This is the default, and the recommended option.
- **If only high-resolution timer (systimer) is used, gettimeofday will** provide time at microsecond resolution. Time will not be preserved when going into deep sleep mode.
- **If only RTC timer is used, timekeeping will continue in** deep sleep, but time will be measured at 6.6(6) microsecond resolution. Also the gettimeofday function itself may take longer to run.
- **If no timers are used, gettimeofday and time functions** return -1 and set errno to ENOSYS.
- **When RTC is used for timekeeping, two RTC_STORE registers are** used to keep time in deep sleep mode.

**Available options:**

- RTC and high-resolution timer (NEWLIB_TIME_SYSCALL_USE_RTC_HRT)
- RTC (NEWLIB_TIME_SYSCALL_USE_RTC)
- High-resolution timer (NEWLIB_TIME_SYSCALL_USE_HRT)
- None (NEWLIB_TIME_SYSCALL_USE_NONE)

**NVS**

Contains:

- **CONFIG_NVS_ENCRYPTION**
- **CONFIG_NVS_COMPATIBLE_PRE_V4_3_ENCRYPTION_FLAG**
- **CONFIG_NVS_ASSERT_ERROR_CHECK**

**CONFIG_NVS_ENCRYPTION**

Enable NVS encryption

*Found in: Component config > NVS*

This option enables encryption for NVS. When enabled, AES-XTS is used to encrypt the complete NVS data, except the page headers. It requires XTS encryption keys to be stored in an encrypted partition. This means enabling flash encryption is a pre-requisite for this feature.

**Default value:**

- Yes (enabled) if **CONFIG_SECURE_FLASH_ENC_ENABLED**

**CONFIG_NVS_COMPATIBLE_PRE_V4_3_ENCRYPTION_FLAG**

NVS partition encrypted flag compatible with ESP-IDF before v4.3

*Found in: Component config > NVS*

Enabling this will ignore “encrypted” flag for NVS partitions. NVS encryption scheme is different than hardware flash encryption and hence it is not recommended to have “encrypted” flag for NVS partitions. This was not being checked in pre v4.3 IDF. Hence, if you have any devices where this flag is kept enabled in partition table then enabling this config will allow to have same behavior as pre v4.3 IDF.
CONFIG_NVS_ASSERT_ERROR_CHECK
Use assertions for error checking

*Found in: Component config > NVS*

This option switches error checking type between assertions (y) or return codes (n).

**Default value:**
- No (disabled)

**OpenThread**  Contains:
- `CONFIG_OPENTHREAD_ENABLED`

CONFIG_OPENTHREAD_ENABLED
OpenThread

*Found in: Component config > OpenThread*

Select this option to enable OpenThread and show the submenu with OpenThread configuration choices.

**Default value:**
- No (disabled)

CONFIG_OPENTHREAD_LOG_LEVEL_DYNAMIC
Enable dynamic log level control

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to enable dynamic log level control for OpenThread

**Default value:**
- Yes (enabled) if `CONFIG_OPENTHREAD_ENABLED`

CONFIG_OPENTHREAD_LOG_LEVEL
OpenThread log verbosity

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select OpenThread log level.

**Available options:**
- No logs (OPENTHREAD_LOG_LEVEL_NONE)
- Error logs (OPENTHREAD_LOG_LEVEL_CRIT)
- Warning logs (OPENTHREAD_LOG_LEVEL_WARN)
- Notice logs (OPENTHREAD_LOG_LEVEL_NOTE)
- Info logs (OPENTHREAD_LOG_LEVEL_INFO)
- Debug logs (OPENTHREAD_LOG_LEVEL_DEBUG)

CONFIG_OPENTHREAD_RADIO_TYPE
Config the Thread radio type

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Configure how OpenThread connects to the 15.4 radio

**Available options:**
- Native 15.4 radio (OPENTHREAD_RADIO_NATIVE)
  Select this to use the native 15.4 radio.
• Connect via UART (OPENTHREAD_RADIO_SPINEL_UART)
  Select this to connect to a Radio Co-Processor via UART.

**CONFIG_OPENTHREAD_DEVICE_TYPE**

Config the Thread device type

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

OpenThread can be configured to different device types (FTD, MTD, Radio)

**Available options:**

- **Full Thread Device (OPENTHREAD_FTD)**
  Select this to enable Full Thread Device which can act as router and leader in a Thread network.

- **Minimal Thread Device (OPENTHREAD_MTD)**
  Select this to enable Minimal Thread Device which can only act as end device in a Thread network. This will reduce the code size of the OpenThread stack.

- **Radio Only Device (OPENTHREAD_RADIO)**
  Select this to enable Radio Only Device which can only forward 15.4 packets to the host. The OpenThread stack will be run on the host and OpenThread will have minimal footprint on the radio only device.

**CONFIG_OPENTHREAD_CLI**

Enable OpenThread Command-Line Interface

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to enable Command-Line Interface in OpenThread.

**Default value:**

- Yes (enabled) if CONFIG_OPENTHREAD_ENABLED

**CONFIG_OPENTHREAD_DIAG**

Enable diag

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to enable diag in OpenThread. This will enable diag mode and a series of diag commands in the OpenThread command line. These commands allow users to manipulate low-level features of the storage and 15.4 radio.

**Default value:**

- Yes (enabled) if CONFIG_OPENTHREAD_ENABLED

**CONFIG_OPENTHREAD_COMMISSIONER**

Enable Commissioner

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to enable commissioner in OpenThread. This will enable the device to act as a commissioner in the Thread network. A commissioner checks the pre-shared key from a joining device with the Thread commissioning protocol and shares the network parameter with the joining device upon success.

**Default value:**

- No (disabled) if CONFIG_OPENTHREAD_ENABLED
CONFIG_OPENTHREAD_JOINER

Enable Joiner

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to enable Joiner in OpenThread. This allows a device to join the Thread network with a pre-shared key using the Thread commissioning protocol.

**Default value:**
- No (disabled) if `CONFIG_OPENTHREAD_ENABLED`

CONFIG_OPENTHREAD_SRP_CLIENT

Enable SRP Client

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to enable SRP Client in OpenThread. This allows a device to register SRP services to SRP Server.

**Default value:**
- No (disabled) if `CONFIG_OPENTHREAD_ENABLED`

CONFIG_OPENTHREAD_BORDER_ROUTER

Enable Border Router

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to enable border router features in OpenThread.

**Default value:**
- No (disabled) if `CONFIG_OPENTHREAD_ENABLED`

CONFIG_OPENTHREAD_NUM_MESSAGE_BUFFERS

The number of openthread message buffers

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

**Range:**
- from 50 to 100 if `CONFIG_OPENTHREAD_ENABLED`

**Default value:**
- 65 if `CONFIG_OPENTHREAD_ENABLED`

CONFIG_OPENTHREAD_DNS64_CLIENT

Use dns64 client

*Found in: Component config > OpenThread > CONFIG_OPENTHREAD_ENABLED*

Select this option to acquire NAT64 address from dns servers.

**Default value:**
- No (disabled) if `CONFIG_OPENTHREAD_ENABLED`

Protocomm Contains:

- `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0`
- `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1`
- `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2`
CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0
Support protocomm security version 0 (no security)

Found in: Component config > Protocomm

Enable support of security version 0. Disabling this option saves some code size. Consult the Enabling protocomm security version section of the Protocomm documentation in ESP-IDF Programming guide for more details.

Default value:
• Yes (enabled)

CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1
Support protocomm security version 1 (Curve25519 key exchange + AES-CTR encryption/decryption)

Found in: Component config > Protocomm

Enable support of security version 1. Disabling this option saves some code size. Consult the Enabling protocomm security version section of the Protocomm documentation in ESP-IDF Programming guide for more details.

Default value:
• Yes (enabled)

CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2
Support protocomm security version 2 (SRP6a-based key exchange + AES-GCM encryption/decryption)

Found in: Component config > Protocomm

Enable support of security version 2. Disabling this option saves some code size. Consult the Enabling protocomm security version section of the Protocomm documentation in ESP-IDF Programming guide for more details.

Default value:
• Yes (enabled)

PThreads Contains:
• CONFIG_PTHREAD_TASK_NAME_DEFAULT
• CONFIG_PTHREAD_TASK_CORE_DEFAULT
• CONFIG_PTHREAD_TASK_PRIO_DEFAULT
• CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT
• CONFIG_PTHREAD_STACK_MIN

CONFIG_PTHREAD_TASK_PRIO_DEFAULT
Default task priority

Found in: Component config > PThreads

Priority used to create new tasks with default pthread parameters.

Range:
• from 0 to 255

Default value:
• 5
**CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT**

Default task stack size

*Found in: Component config > PThreads*

Stack size used to create new tasks with default pthread parameters.

**Default value:**
- 3072

**CONFIG_PTHREAD_STACK_MIN**

Minimum allowed pthread stack size

*Found in: Component config > PThreads*

Minimum allowed pthread stack size set in attributes passed to pthread_create

**Default value:**
- 768

**CONFIG_PTHREAD_TASK_CORE_DEFAULT**

Default pthread core affinity

*Found in: Component config > PThreads*

The default core to which pthreads are pinned.

**Available options:**
- No affinity (PTHREAD_DEFAULT_CORE_NO_AFFINITY)
- Core 0 (PTHREAD_DEFAULT_CORE_0)
- Core 1 (PTHREAD_DEFAULT_CORE_1)

**CONFIG_PTHREAD_TASK_NAME_DEFAULT**

Default name of pthreads

*Found in: Component config > PThreads*

The default name of pthreads.

**Default value:**
- “pthread”

**SPI Flash driver**  
Contains:
- Auto-detect flash chips
- CONFIG_SPI_FLASH_BYPASS_BLOCK_ERASE
- CONFIG_SPI_FLASH_ENABLE_ENCRYPTED_READ_WRITE
- CONFIG_SPI_FLASH_ENABLE_COUNTERS
- CONFIG_SPI_FLASH_ROM_DRIVER_PATCH
- CONFIG_SPI_FLASH_YIELD_DURING_ERASE
- CONFIG_SPI_FLASH_CHECK_ERASE_TIMEOUT_DISABLED
- CONFIG_SPI_FLASH_WRITE_CHUNK_SIZE
- CONFIG_SPI_FLASH_OVERRIDE_ERASE_CHIP_DRIVER_LIST
- CONFIG_SPI_FLASH_SIZE_OVERRIDE
- SPI Flash behavior when brownout
- CONFIG_SPI_FLASH_SHARE_SPI1_BUS
- CONFIG_SPI_FLASH_ROM_IMPL
- CONFIG_SPI_FLASH_VERIFY_WRITE
- CONFIG_SPI_FLASH_DANGEROUS_WRITE
CONFIG_SPI_FLASH_VERIFY_WRITE

Verify SPI flash writes

*Found in: Component config > SPI Flash driver*

If this option is enabled, any time SPI flash is written then the data will be read back and verified. This can catch hardware problems with SPI flash, or flash which was not erased before verification.

**Default value:**
- No (disabled)

CONFIG_SPI_FLASH_LOG_FAILED_WRITE

Log errors if verification fails

*Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_VERIFY_WRITE*

If this option is enabled, if SPI flash write verification fails then a log error line will be written with the address, expected & actual values. This can be useful when debugging hardware SPI flash problems.

**Default value:**
- No (disabled) if `CONFIG_SPI_FLASH_VERIFY_WRITE`

CONFIG_SPI_FLASH_WARN_SETTING_ZERO_TO_ONE

Log warning if writing zero bits to ones

*Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_VERIFY_WRITE*

If this option is enabled, any SPI flash write which tries to set zero bits in the flash to ones will log a warning. Such writes will not result in the requested data appearing identically in flash once written, as SPI NOR flash can only set bits to one when an entire sector is erased. After erasing, individual bits can only be written from one to zero.

Note that some software (such as SPIFFS) which is aware of SPI NOR flash may write one bits as an optimisation, relying on the data in flash becoming a bitwise AND of the new data and any existing data. Such software will log spurious warnings if this option is enabled.

**Default value:**
- No (disabled) if `CONFIG_SPI_FLASH_VERIFY_WRITE`

CONFIG_SPI_FLASH_ENABLE_COUNTERS

Enable operation counters

*Found in: Component config > SPI Flash driver*

This option enables the following APIs:
- `spi_flash_reset_counters`
- `spi_flash_dump_counters`
- `spi_flash_get_counters`

These APIs may be used to collect performance data for spi flash APIs and to help understand behaviour of libraries which use SPI flash.

**Default value:**
- 0
CONFIG_SPI_FLASH_ROM_DRIVER_PATCH

Enable SPI flash ROM driver patched functions

*Found in: Component config > SPI Flash driver*

Enable this flag to use patched versions of SPI flash ROM driver functions. This option should be enabled, if any one of the following is true: (1) need to write to flash on ESP32-D2WD; (2) main SPI flash is connected to non-default pins; (3) main SPI flash chip is manufactured by ISSI.

**Default value:**
- Yes (enabled)

CONFIG_SPI_FLASH_ROM_IMPL

Use esp_flash implementation in ROM

*Found in: Component config > SPI Flash driver*

Enable this flag to use new SPI flash driver functions from ROM instead of ESP-IDF.

If keeping this as “n” in your project, you will have less free IRAM. But you can use all of our flash features.

If making this as “y” in your project, you will increase free IRAM. But you may miss out on some flash features and support for new flash chips.

Currently the ROM cannot support the following features:
- SPI_FLASH_AUTO_SUSPEND (C3, S3)

**Default value:**
- No (disabled) if ESP_ROM_HAS_SPI_FLASH

CONFIG_SPI_FLASH_DANGEROUS_WRITE

Writing to dangerous flash regions

*Found in: Component config > SPI Flash driver*

SPI flash APIs can optionally abort or return a failure code if erasing or writing addresses that fall at the beginning of flash (covering the bootloader and partition table) or that overlap the app partition that contains the running app.

It is not recommended to ever write to these regions from an IDF app, and this check prevents logic errors or corrupted firmware memory from damaging these regions.

Note that this feature *does not* check calls to the esp_rom_xxx SPI flash ROM functions. These functions should not be called directly from IDF applications.

**Available options:**
- Aborts (SPI_FLASH_DANGEROUS_WRITE_ABORTS)
- Fails (SPI_FLASH_DANGEROUS_WRITE_FAILS)
- Allowed (SPI_FLASH_DANGEROUS_WRITE_ALLOWED)

CONFIG_SPI_FLASH_SHARE_SPI1_BUS

Support other devices attached to SPI1 bus

*Found in: Component config > SPI Flash driver*

Each SPI bus needs a lock for arbitration among devices. This allows multiple devices on a same bus, but may reduce the speed of esp_flash driver access to the main flash chip.

If you only need to use esp_flash driver to access the main flash chip, disable this option, and the lock will be bypassed on SPI1 bus. Otherwise if extra devices are needed to attach to SPI1 bus, enable this option.
**CONFIG_SPI_FLASH_BYPASS_BLOCK_ERASE**

Bypass a block erase and always do sector erase

*Found in: Component config > SPI Flash driver*

Some flash chips can have very high “max” erase times, especially for block erase (32KB or 64KB). This option allows to bypass “block erase” and always do sector erase commands. This will be much slower overall in most cases, but improves latency for other code to run.

*Default value:*
  - No (disabled)

**CONFIG_SPI_FLASH_YIELD_DURING_ERASE**

Enables yield operation during flash erase

*Found in: Component config > SPI Flash driver*

This allows to yield the CPUs between erase commands. Prevents starvation of other tasks. Please use this configuration together with SPI\_FLASH\_ERASE\_YIELD\_DURATION\_MS and SPI\_FLASH\_ERASE\_YIELD\_TICKS after carefully checking flash datasheet to avoid a watchdog timeout. For more information, please check SPI Flash API reference documentation under section OS Function.

*Default value:*
  - Yes (enabled)

**CONFIG_SPI_FLASH_ERASE_YIELD_DURATION_MS**

Duration of erasing to yield CPUs (ms)

*Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_YIELD_DURING_ERASE*

If a duration of one erase command is large then it will yield CPUs after finishing a current command.

*Default value:*
  - 20

**CONFIG_SPI_FLASH_ERASE_YIELD_TICKS**

CPU release time (tick) for an erase operation

*Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_YIELD_DURING_ERASE*

Defines how many ticks will be before returning to continue a erasing.

*Default value:*
  - 1

**CONFIG_SPI_FLASH_WRITE_CHUNK_SIZE**

Flash write chunk size

*Found in: Component config > SPI Flash driver*

Flash write is broken down in terms of multiple (smaller) write operations. This configuration options helps to set individual write chunk size, smaller value here ensures that cache (and non-IRAM resident interrupts) remains disabled for shorter duration.

*Range:*
  - from 256 to 8192
Default value:
- 8192

**CONFIG_SPI_FLASH_SIZE_OVERRIDE**
Override flash size in bootloader header by ESPTOOLPY_FLASHSIZE

*Found in: Component config > SPI Flash driver*

SPI Flash driver uses the flash size configured in bootloader header by default. Enable this option to override flash size with latest ESPTOOLPY_FLASHSIZE value from the app header if the size in the bootloader header is incorrect.

Default value:
- No (disabled)

**CONFIG_SPI_FLASH_CHECK_ERASE_TIMEOUT_DISABLED**
Flash timeout checkout disabled

*Found in: Component config > SPI Flash driver*

This option is helpful if you are using a flash chip whose timeout is quite large or unpredictable.

Default value:
- No (disabled)

**CONFIG_SPI_FLASH_OVERRIDE_CHIP_DRIVER_LIST**
Override default chip driver list

*Found in: Component config > SPI Flash driver*

This option allows the chip driver list to be customized, instead of using the default list provided by ESP-IDF.

When this option is enabled, the default list is no longer compiled or linked. Instead, the default_registered_chips structure must be provided by the user.

See example: custom_chip_driver under examples/storage for more details.

Default value:
- No (disabled)

**SPI Flash behavior when brownout**
Contains:
- **CONFIG_SPI_FLASH_BROWNOUT_RESET_XMC**

**CONFIG_SPI_FLASH_BROWNOUT_RESET_XMC**
Enable sending reset when brownout for XMC flash chips

*Found in: Component config > SPI Flash driver > SPI Flash behavior when brownout*

When this option is selected, the patch will be enabled for XMC. Follow the recommended flow by XMC for better stability.

**DO NOT DISABLE UNLESS YOU KNOW WHAT YOU ARE DOING.**

Default value:
- Yes (enabled)
Auto-detect flash chips Contains:

- CONFIG_SPI_FLASH_SUPPORT_BOYA_CHIP
- CONFIG_SPI_FLASH_SUPPORT_GD_CHIP
- CONFIG_SPI_FLASH_SUPPORT_ISSI_CHIP
- CONFIG_SPI_FLASH_SUPPORT_MXIC_CHIP
- CONFIG_SPI_FLASH_SUPPORT_TH_CHIP
- CONFIG_SPI_FLASH_SUPPORT_WINBOND_CHIP

CONFIG_SPI_FLASH_SUPPORT_ISSI_CHIP

ISSI

*Found in: Component config > SPI Flash driver > Auto-detect flash chips*

Enable this to support auto detection of ISSI chips if chip vendor not directly given by chip\_drv member of the chip struct. This adds support for variant chips, however will extend detecting time.

**Default value:**
- Yes (enabled)

CONFIG_SPI_FLASH_SUPPORT_MXIC_CHIP

MXIC

*Found in: Component config > SPI Flash driver > Auto-detect flash chips*

Enable this to support auto detection of MXIC chips if chip vendor not directly given by chip\_drv member of the chip struct. This adds support for variant chips, however will extend detecting time.

**Default value:**
- Yes (enabled)

CONFIG_SPI_FLASH_SUPPORT_GD_CHIP

GigaDevice

*Found in: Component config > SPI Flash driver > Auto-detect flash chips*

Enable this to support auto detection of GD (GigaDevice) chips if chip vendor not directly given by chip\_drv member of the chip struct. If you are using Wrover modules, please don’t disable this, otherwise your flash may not work in 4-bit mode.

This adds support for variant chips, however will extend detecting time and image size. Note that the default chip driver supports the GD chips with product ID 60H.

**Default value:**
- Yes (enabled)

CONFIG_SPI_FLASH_SUPPORT_WINBOND_CHIP

Winbond

*Found in: Component config > SPI Flash driver > Auto-detect flash chips*

Enable this to support auto detection of Winbond chips if chip vendor not directly given by chip\_drv member of the chip struct. This adds support for variant chips, however will extend detecting time.

**Default value:**
- Yes (enabled)
CONFIG_SPI_FLASH_SUPPORT_BOYA_CHIP

BOYA

*Found in: Component config > SPI Flash driver > Auto-detect flash chips*

Enable this to support auto detection of BOYA chips if chip vendor not directly given by `chip_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

**Default value:**
- No (disabled)
- Yes (enabled)

CONFIG_SPI_FLASH_SUPPORT_TH_CHIP

TH

*Found in: Component config > SPI Flash driver > Auto-detect flash chips*

Enable this to support auto detection of TH chips if chip vendor not directly given by `chip_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

**Default value:**
- No (disabled)
- Yes (enabled)

CONFIG_SPI_FLASH_ENABLE_ENCRYPTED_READ_WRITE

Enable encrypted partition read/write operations

*Found in: Component config > SPI Flash driver*

This option enables flash read/write operations to encrypted partition/s. This option is kept enabled irrespective of state of flash encryption feature. However, in case application is not using flash encryption feature and is in need of some additional memory from IRAM region (~1KB) then this config can be disabled.

**Default value:**
- Yes (enabled)

SPIFFS Configuration  Contains:

- Debug Configuration
- CONFIG_SPIFFS_USE_MAGIC
- CONFIG_SPIFFS_GC_STATS
- CONFIG_SPIFFS_PAGE_CHECK
- CONFIG_SPIFFS_FOLLOW_SYMLINKS
- CONFIG_SPIFFS_MAX_PARTITIONS
- CONFIG_SPIFFS_USE_MTIME
- CONFIG_SPIFFS_GC_MAX_RUNS
- CONFIG_SPIFFS_OBJ_NAME_LEN
- CONFIG_SPIFFS_META_LENGTH
- SPIFFS Cache Configuration
- CONFIG_SPIFFS_PAGE_SIZE
- CONFIG_SPIFFS_MTIME_WIDE_64_BITS

CONFIG_SPIFFS_MAX_PARTITIONS

Maximum Number of Partitions

*Found in: Component config > SPIFFS Configuration*

Define maximum number of partitions that can be mounted.
Chapter 2. API

**SPIFFS Cache Configuration**
Contains:

- `CONFIG_SPIFFS_CACHE`

**CONFIG_SPIFFS_CACHE**
Enable SPIFFS Cache

*Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration*

Enables/disables memory read caching of nucleus file system operations.

**Default value:**
- Yes (enabled)

**CONFIG_SPIFFS_CACHE_WR**
Enable SPIFFS Write Caching

*Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration > CONFIG_SPIFFS_CACHE*

Enables memory write caching for file descriptors in hydrogen.

**Default value:**
- Yes (enabled)

**CONFIG_SPIFFS_CACHE_STATS**
Enable SPIFFS Cache Statistics

*Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration > CONFIG_SPIFFS_CACHE*

Enable/disable statistics on caching. Debug/test purpose only.

**Default value:**
- No (disabled)

**CONFIG_SPIFFS_PAGE_CHECK**
Enable SPIFFS Page Check

*Found in: Component config > SPIFFS Configuration*

Always check header of each accessed page to ensure consistent state. If enabled it will increase number of reads from flash, especially if cache is disabled.

**Default value:**
- Yes (enabled)

**CONFIG_SPIFFS_GC_MAX_RUNS**
Set Maximum GC Runs

*Found in: Component config > SPIFFS Configuration*

Define maximum number of GC runs to perform to reach desired free pages.
Range:
- from 1 to 10000

Default value:
- 10

**CONFIG_SPIFFS_GC_STATS**

Enable SPIFFS GC Statistics

*Found in: Component config > SPIFFS Configuration*

Enable/disable statistics on gc. Debug/test purpose only.

Default value:
- No (disabled)

**CONFIG_SPIFFS_PAGE_SIZE**

SPIFFS logical page size

*Found in: Component config > SPIFFS Configuration*

Logical page size of SPIFFS partition, in bytes. Must be multiple of flash page size (which is usually 256 bytes). Larger page sizes reduce overhead when storing large files, and improve filesystem performance when reading large files. Smaller page sizes reduce overhead when storing small (< page size) files.

Range:
- from 256 to 1024

Default value:
- 256

**CONFIG_SPIFFS_OBJ_NAME_LEN**

Set SPIFFS Maximum Name Length

*Found in: Component config > SPIFFS Configuration*

Object name maximum length. Note that this length include the zero-termination character, meaning maximum string of characters can at most be SPIFFS_OBJ_NAME_LEN - 1.

SPIFFS_OBJ_NAME_LEN + SPIFFS_META_LENGTH should not exceed SPIFFS_PAGE_SIZE - 64.

Range:
- from 1 to 256

Default value:
- 32

**CONFIG_SPIFFS_FOLLOW_SYMLINKS**

Enable symbolic links for image creation

*Found in: Component config > SPIFFS Configuration*

If this option is enabled, symbolic links are taken into account during partition image creation.

Default value:
- No (disabled)
Chapter 2. API

CONFIG_SPIFFS_USE_MAGIC

Enable SPIFFS Filesystem Magic

*Found in: Component config > SPIFFS Configuration*

Enable this to have an identifiable spiffs filesystem. This will look for a magic in all sectors to determine if this is a valid spiffs system or not at mount time.

**Default value:**

- Yes (enabled)

CONFIG_SPIFFS_USE_MAGIC_LENGTH

Enable SPIFFS Filesystem Length Magic

*Found in: Component config > SPIFFS Configuration > CONFIG_SPIFFS_USE_MAGIC*

If this option is enabled, the magic will also be dependent on the length of the filesystem. For example, a filesystem configured and formatted for 4 megabytes will not be accepted for mounting with a configuration defining the filesystem as 2 megabytes.

**Default value:**

- Yes (enabled)

CONFIG_SPIFFS_META_LENGTH

Size of per-file metadata field

*Found in: Component config > SPIFFS Configuration*

This option sets the number of extra bytes stored in the file header. These bytes can be used in an application-specific manner. Set this to at least 4 bytes to enable support for saving file modification time.

SPIFFS_OBJ_NAME_LEN + SPIFFS_META_LENGTH should not exceed SPIFFS_PAGE_SIZE - 64.

**Default value:**

- 4

CONFIG_SPIFFS_USE_MTIME

Save file modification time

*Found in: Component config > SPIFFS Configuration*

If enabled, then the first 4 bytes of per-file metadata will be used to store file modification time (mtime), accessible through stat/fstat functions. Modification time is updated when the file is opened.

**Default value:**

- Yes (enabled)

CONFIG_SPIFFS_MTIME_WIDE_64_BITS

The time field occupies 64 bits in the image instead of 32 bits

*Found in: Component config > SPIFFS Configuration*

If this option is not set, the time field is 32 bits (up to 2106 year), otherwise it is 64 bits and make sure it matches SPIFFS_META_LENGTH. If the chip already has the spiffs image with the time field = 32 bits then this option cannot be applied in this case. Erase it first before using this option. To resolve the Y2K38 problem for the spiffs, use a toolchain with 64-bit time_t support.

**Default value:**

- No (disabled) if CONFIG_SPIFFS_META_LENGTH >= 8
Chapter 2. API

Debug Configuration

Contains:

- `CONFIG_SPIFFS_DBG`
- `CONFIG_SPIFFS_API_DBG`
- `CONFIG_SPIFFS_CACHE_DBG`
- `CONFIG_SPIFFS_CHECK_DBG`
- `CONFIG_SPIFFS_TEST_VISUALISATION`
- `CONFIG_SPIFFS_GC_DBG`

`CONFIG_SPIFFS_DBG`

Enable general SPIFFS debug

*Found in: Component config > SPIFFS Configuration > Debug Configuration*

Enabling this option will print general debug messages to the console.

**Default value:**

- No (disabled)

`CONFIG_SPIFFS_API_DBG`

Enable SPIFFS API debug

*Found in: Component config > SPIFFS Configuration > Debug Configuration*

Enabling this option will print API debug messages to the console.

**Default value:**

- No (disabled)

`CONFIG_SPIFFS_GC_DBG`

Enable SPIFFS Garbage Cleaner debug

*Found in: Component config > SPIFFS Configuration > Debug Configuration*

Enabling this option will print GC debug messages to the console.

**Default value:**

- No (disabled)

`CONFIG_SPIFFS_CACHE_DBG`

Enable SPIFFS Cache debug

*Found in: Component config > SPIFFS Configuration > Debug Configuration*

Enabling this option will print cache debug messages to the console.

**Default value:**

- No (disabled)

`CONFIG_SPIFFS_CHECK_DBG`

Enable SPIFFS Filesystem Check debug

*Found in: Component config > SPIFFS Configuration > Debug Configuration*

Enabling this option will print Filesystem Check debug messages to the console.

**Default value:**

- No (disabled)
CONFIG_SPIFFS_TEST_VISUALISATION

Enable SPIFFS Filesystem Visualization

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enable this option to enable SPIFFS_vis function in the API.

Default value:
• No (disabled)

TCP Transport Contains:
• Websocket

Websocket Contains:
• CONFIG_WS_TRANSPORT

CONFIG_WS_TRANSPORT

Enable Websocket Transport

Found in: Component config > TCP Transport > Websocket

Enable support for creating websocket transport.

Default value:
• Yes (enabled)

CONFIG_WS_BUFFER_SIZE

Websocket transport buffer size

Found in: Component config > TCP Transport > Websocket > CONFIG_WS_TRANSPORT

Size of the buffer used for constructing the HTTP Upgrade request during connect

Default value:
• 1024

CONFIG_WS_DYNAMIC_BUFFER

Using dynamic websocket transport buffer

Found in: Component config > TCP Transport > Websocket > CONFIG_WS_TRANSPORT

If enable this option, websocket transport buffer will be freed after connection succeed to save more heap.

Default value:
• No (disabled)

Ultra Low Power (ULP) Co-processor Contains:
• CONFIG_ULP_COPROC_ENABLED
• ULP RISC-V Settings
Chapter 2. API

### CONFIG_ULP_COPROC_ENABLED

Enable Ultra Low Power (ULP) Co-processor

*Found in: Component config > Ultra Low Power (ULP) Co-processor*

Enable this feature if you plan to use the ULP Co-processor. Once this option is enabled, further ULP co-processor configuration will appear in the menu.

**Default value:**
- No (disabled)

### CONFIG_ULP_COPROC_TYPE

ULP Co-processor type

*Found in: Component config > Ultra Low Power (ULP) Co-processor > CONFIG_ULP_COPROC_ENABLED*

Choose the ULP Coprocessor type: ULP FSM (Finite State Machine) or ULP RISC-V. Please note that ESP32 only supports ULP FSM.

**Available options:**
- ULP FSM (Finite State Machine) (ULP_COPROC_TYPE_FSM)
- ULP RISC-V (ULP_COPROC_TYPE_RISCV)

### CONFIG_ULP_COPROC_RESERVE_MEM

RTC Slow memory reserved for coprocessor

*Found in: Component config > Ultra Low Power (ULP) Co-processor > CONFIG_ULP_COPROC_ENABLED*

Bytes of memory to reserve for ULP Co-processor firmware & data. Data is reserved at the beginning of RTC slow memory.

**Range:**
- from 32 to 8176 if `CONFIG_ULP_COPROC_ENABLED`

**Default value:**
- 512 if `CONFIG_ULP_COPROC_ENABLED`

### ULP RISC-V Settings

#### Unity unit testing library

Contains:

- `CONFIG.Unity_ENABLE_COLOR`
- `CONFIG.Unity_ENABLE.IDF_TEST_RUNNER`
- `CONFIG.Unity_ENABLE_FIXTURE`
- `CONFIG.Unity_ENABLE_BACKTRACE_ON_FAIL`
- `CONFIG.Unity_ENABLE.64BIT`
- `CONFIG.Unity_ENABLE_DOUBLE`
- `CONFIG.Unity_ENABLE_FLOAT`

### CONFIG.Unity_ENABLE_FLOAT

Support for float type

*Found in: Component config > Unity unit testing library*

If not set, assertions on float arguments will not be available.

**Default value:**
- Yes (enabled)
CONFIG_UNITY_ENABLE_DOUBLE
Support for double type

*Found in: Component config > Unity unit testing library*

If not set, assertions on double arguments will not be available.

**Default value:**
- Yes (enabled)

CONFIG_UNITY_ENABLE_64BIT
Support for 64-bit integer types

*Found in: Component config > Unity unit testing library*

If not set, assertions on 64-bit integer types will always fail. If this feature is enabled, take care not to pass pointers (which are 32 bit) to UNITY_ASSERT_EQUAL, as that will cause pointer-to-int-cast warnings.

**Default value:**
- No (disabled)

CONFIG_UNITY_ENABLE_COLOR
Colorize test output

*Found in: Component config > Unity unit testing library*

If set, Unity will colorize test results using console escape sequences.

**Default value:**
- No (disabled)

CONFIG_UNITY_ENABLE_IDF_TEST_RUNNER
Include ESP-IDF test registration/running helpers

*Found in: Component config > Unity unit testing library*

If set, then the following features will be available:

- TEST_CASE macro which performs automatic registration of test functions
- Functions to run registered test functions: unity_run_all_tests, unity_run_tests_with_filter, unity_run_single_test_by_name.
- Interactive menu which lists test cases and allows choosing the tests to be run, available via unity_run_menu function.

Disable if a different test registration mechanism is used.

**Default value:**
- Yes (enabled)

CONFIG_UNITY_ENABLE_FIXTURE
Include Unity test fixture

*Found in: Component config > Unity unit testing library*

If set, unity_fixture.h header file and associated source files are part of the build. These provide an optional set of macros and functions to implement test groups.

**Default value:**
- No (disabled)
CONFIG_UNITY_ENABLE_BACKTRACE_ON_FAIL
Print a backtrace when a unit test fails

*Found in: Component config > Unity unit testing library*

If set, the unity framework will print the backtrace information before jumping back to the test menu. The jumping is usually occurs in assert functions such as TEST_ASSERT, TEST_FAIL etc.

**Default value:**
- No (disabled)

Virtual file system  Contains:
- **CONFIG_VFS_SUPPORT_IO**

CONFIG_VFS_SUPPORT_IO
Provide basic I/O functions

*Found in: Component config > Virtual file system*

If enabled, the following functions are provided by the VFS component:
- open, close, read, write, pread, pwrite, lseek, fstat, fsync, ioctl, fcntl

Filesystem drivers can then be registered to handle these functions for specific paths.

Disabling this option can save memory when the support for these functions is not required.

Note that the following functions can still be used with socket file descriptors when this option is disabled:
- close, read, write, ioctl, fcntl.

**Default value:**
- Yes (enabled)

CONFIG_VFS_SUPPORT_DIR
Provide directory related functions

*Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO*

If enabled, the following functions are provided by the VFS component:
- stat, link, unlink, rename, utime, access, truncate, rmdir, mkdir, opendir, closedir, readdir, readdir_r, seekdir, telldir, rewinddir

Filesystem drivers can then be registered to handle these functions for specific paths.

Disabling this option can save memory when the support for these functions is not required.

**Default value:**
- Yes (enabled)

CONFIG_VFS_SUPPORT_SELECT
Provide select function

*Found in: Component config > Virtual file system > CONFIG_VFS_SUPPORT_IO*

If enabled, select function is provided by the VFS component, and can be used on peripheral file descriptors (such as UART) and sockets at the same time.

If disabled, the default select implementation will be provided by LWIP for sockets only.

Disabling this option can reduce code size if support for “select” on UART file descriptors is not required.
**Default value:**
- Yes (enabled) if \( \text{CONFIG\_VFS\_SUPPORT\_IO} \) && \( \text{CONFIG\_LWIP\_USE\_ONLY\_LWIP\_SELECT} \)

**CONFIG\_VFS\_SUPPRESS\_SELECT\_DEBUG\_OUTPUT**

Suppress select() related debug outputs

*Found in:* Component config > Virtual file system > CONFIG\_VFS\_SUPPORT\_IO > CONFIG\_VFS\_SUPPORT\_SELECT

Select() related functions might produce an unconveniently lot of debug outputs when one sets the default log level to DEBUG or higher. It is possible to suppress these debug outputs by enabling this option.

**Default value:**
- Yes (enabled)

**CONFIG\_VFS\_SUPPORT\_TERMIOS**

Provide termios.h functions

*Found in:* Component config > Virtual file system > CONFIG\_VFS\_SUPPORT\_IO

Disabling this option can save memory when the support for termios.h is not required.

**Default value:**
- Yes (enabled)

**Host File System I/O (Semihosting)**

Contains:
- \( \text{CONFIG\_VFS\_SEMIHOSTFS\_MAX\_MOUNT\_POINTS} \)

**CONFIG\_VFS\_SEMIHOSTFS\_MAX\_MOUNT\_POINTS**

Host FS: Maximum number of the host filesystem mount points

*Found in:* Component config > Virtual file system > CONFIG\_VFS\_SUPPORT\_IO > Host File System I/O (Semihosting)

Define maximum number of host filesystem mount points.

**Default value:**
- 1

**Wear Levelling**

Contains:
- \( \text{CONFIG\_WL\_SECTOR\_MODE} \)
- \( \text{CONFIG\_WL\_SECTOR\_SIZE} \)

**CONFIG\_WL\_SECTOR\_SIZE**

Wear Levelling library sector size

*Found in:* Component config > Wear Levelling

Sector size used by wear levelling library. You can set default sector size or size that will fit to the flash device sector size.

With sector size set to 4096 bytes, wear levelling library is more efficient. However if FAT filesystem is used on top of wear levelling library, it will need more temporary storage: 4096 bytes for each mounted filesystem and 4096 bytes for each opened file.
With sector size set to 512 bytes, wear levelling library will perform more operations with flash memory, but less RAM will be used by FAT filesystem library (512 bytes for the filesystem and 512 bytes for each file opened).

**Available options:**

- 512 (WL_SECTOR_SIZE_512)
- 4096 (WL_SECTOR_SIZE_4096)

**CONFIG_WL_SECTOR_MODE**

Sector store mode

*Found in: Component config > Wear Levelling*

Specify the mode to store data into flash:

- In Performance mode a data will be stored to the RAM and then stored back to the flash. Compared to the Safety mode, this operation is faster, but if power will be lost when erase sector operation is in progress, then the data from complete flash device sector will be lost.
- In Safety mode data from complete flash device sector will be read from flash, modified, and then stored back to flash. Compared to the Performance mode, this operation is slower, but if power is lost during erase sector operation, then the data from full flash device sector will not be lost.

**Available options:**

- Performance (WL_SECTOR_MODE_PERF)
- Safety (WL_SECTOR_MODE_SAFE)

**Wi-Fi Provisioning Manager**

Contains:

- `CONFIG_WIFI_PROV_BLE_BONDING`
- `CONFIG_WIFI_PROV_BLE_SEC_CONN`
- `CONFIG_WIFI_PROV_BLE_FORCE_ENCRYPTION`
- `CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV`
- `CONFIG_WIFI_PROV_SCAN_MAX_ENTRIES`
- `CONFIG_WIFI_PROV_AUTOSTOP_TIMEOUT`

**CONFIG_WIFI_PROV_SCAN_MAX_ENTRIES**

Max Wi-Fi Scan Result Entries

*Found in: Component config > Wi-Fi Provisioning Manager*

This sets the maximum number of entries of Wi-Fi scan results that will be kept by the provisioning manager

**Range:**

- from 1 to 255

**Default value:**

- 16

**CONFIG_WIFI_PROV_AUTOSTOP_TIMEOUT**

Provisioning auto-stop timeout

*Found in: Component config > Wi-Fi Provisioning Manager*

Time (in seconds) after which the Wi-Fi provisioning manager will auto-stop after connecting to a Wi-Fi network successfully.

**Range:**

- from 5 to 600

**Default value:**

- 30
CONFIG_WIFI_PROV_BLE_BONDING

Enable BLE bonding

*Found in: Component config > Wi-Fi Provisioning Manager*

This option is applicable only when provisioning transport is BLE.

CONFIG_WIFI_PROV_BLE_SEC_CONN

Enable BLE Secure connection flag

*Found in: Component config > Wi-Fi Provisioning Manager*

Used to enable Secure connection support when provisioning transport is BLE.

**Default value:**
- Yes (enabled) if BT_NIMBLE_ENABLED

CONFIG_WIFI_PROV_BLE_FORCE_ENCRYPTION

Force Link Encryption during characteristic Read / Write

*Found in: Component config > Wi-Fi Provisioning Manager*

Used to enforce link encryption when attempting to read / write characteristic

CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV

Keep BT on after provisioning is done

*Found in: Component config > Wi-Fi Provisioning Manager*

CONFIG_WIFI_PROV_DISCONNECT_AFTER_PROV

Terminate connection after provisioning is done

*Found in: Component config > Wi-Fi Provisioning Manager > CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV*

**Default value:**
- Yes (enabled) if CONFIG_WIFI_PROV_KEEP_BLE_ON_AFTER_PROV

**Supplicant**  Contains:

- CONFIG_WPA_TESTING_OPTIONS
- CONFIG_WPA_WPS_SOFTAP_REGISTRAR
- CONFIG_WPA_11KV_SUPPORT
- CONFIG_WPA_11R_SUPPORT
- CONFIG_WPA_DPP_SUPPORT
- CONFIG_WPA_MBO_SUPPORT
- CONFIG_WPA_SUITE_B_192
- CONFIG_WPA_WAPI_PSK
- CONFIG_WPA_DEBUG_PRINT
- CONFIG_WPA_WPS_STRICT
- CONFIG_WPA_MBEDTLS_CRYPTO
CONFIG_WPA_MBEDTLS_CRYPTO

Use MbedTLS crypto APIs

*Found in: Component config > Supplicant*

Select this option to use MbedTLS crypto APIs which utilize hardware acceleration.

**Default value:**
- Yes (enabled)

CONFIG_WPA_MBEDTLS_TLS_CLIENT

Use MbedTLS TLS client for WiFi Enterprise connection

*Found in: Component config > Supplicant > CONFIG_WPA_MBEDTLS_CRYPTO*

Select this option to use MbedTLS TLS client for WPA2 enterprise connection. Please note that from MbedTLS-3.0 onwards, MbedTLS does not support SSL-3.0, TLS-v1.0, TLS-v1.1 versions. Incase your server is using one of these version, it is advisable to update your server. Please disable this option for compatibility with older TLS versions.

**Default value:**
- Yes (enabled)

CONFIG_WPA_WAPI_PSK

Enable WAPI PSK support

*Found in: Component config > Supplicant*

Select this option to enable WAPI-PSK which is a Chinese National Standard Encryption for Wireless LANs (GB 15629.11-2003).

**Default value:**
- No (disabled)

CONFIG_WPA_SUITE_B_192

Enable NSA suite B support with 192 bit key

*Found in: Component config > Supplicant*

Select this option to enable 192 bit NSA suite-B. This is necessary to support WPA3 192 bit security.

**Default value:**
- No (disabled)

CONFIG_WPA_DEBUG_PRINT

Print debug messages from WPA Supplicant

*Found in: Component config > Supplicant*

Select this option to print logging information from WPA supplicant, this includes handshake information and key hex dumps depending on the project logging level.

Enabling this could increase the build size ~60kb depending on the project logging level.

**Default value:**
- No (disabled)
CONFIG_WPA_TESTING_OPTIONS

Add DPP testing code

*Found in: Component config > Supplicant*

Select this to enable unity test for DPP.

**Default value:**
- No (disabled)

CONFIG_WPA_WPS STRICT

Strictly validate all WPS attributes

*Found in: Component config > Supplicant*

Select this option to enable validate each WPS attribute rigorously. Disabling this add the workarounds with various APs. Enabling this may cause inter operability issues with some APs.

**Default value:**
- No (disabled)

CONFIG_WPA_11KV_SUPPORT

Enable 802.11k, 802.11v APIs Support

*Found in: Component config > Supplicant*

Select this option to enable 802.11k 802.11v APIs (RRM and BTM support). Only APIs which are helpful for network assisted roaming are supported for now. Enable this option with BTM and RRM enabled in sta config to make device ready for network assisted roaming. BTM: BSS transition management enables an AP to request a station to transition to a specific AP, or to indicate to a station a set of preferred APs. RRM: Radio measurements enable STAs to understand the radio environment, it enables STAs to observe and gather data on radio link performance and on the radio environment. Current implementation adds beacon report, link measurement, neighbor report.

**Default value:**
- No (disabled)

CONFIG_WPA_SCAN_CACHE

Keep scan results in cache

*Found in: Component config > Supplicant > CONFIG_WPA_11KV_SUPPORT*

Keep scan results in cache, if not enabled, those will be flushed immediately.

**Default value:**
- No (disabled) if CONFIG_WPA_11KV_SUPPORT

CONFIG_WPA_MBO_SUPPORT

Enable Multi Band Operation Certification Support

*Found in: Component config > Supplicant*

Select this option to enable WiFi Multiband operation certification support.

**Default value:**
- No (disabled)
CONFIG_WPA_DPP_SUPPORT
Enable DPP support

*Found in:* Component config > Supplicant

Select this option to enable WiFi Easy Connect Support.

**Default value:**
- No (disabled)

CONFIG_WPA_11R_SUPPORT
Enable 802.11R (Fast Transition) Support

*Found in:* Component config > Supplicant

Select this option to enable WiFi Fast Transition Support.

**Default value:**
- No (disabled)

CONFIG_WPA_WPS_SOFTAP.REGISTRAR
Add WPS Registrar support in SoftAP mode

*Found in:* Component config > Supplicant

Select this option to enable WPS registrar support in softAP mode.

**Default value:**
- No (disabled)

**Deprecated options and their replacements**

- CONFIG_A2DP_ENABLE (*CONFIG_BT_A2DP_ENABLE*)
- CONFIG_A2D_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_A2D_TRACE_LEVEL*)
  - CONFIG_A2D_TRACE_LEVEL_NONE
  - CONFIG_A2D_TRACE_LEVEL_ERROR
  - CONFIG_A2D_TRACE_LEVEL_WARNING
  - CONFIG_A2D_TRACE_LEVEL_API
  - CONFIG_A2D_TRACE_LEVEL_EVENT
  - CONFIG_A2D_TRACE_LEVEL_DEBUG
  - CONFIG_A2D_TRACE_LEVEL_VERBOSE
- CONFIG_ADC2_DISABLE_DAC (*CONFIG_ADC_DISABLE_DAC*)
- CONFIG_APP_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_APP_TRACE_LEVEL*)
  - CONFIG_APP_TRACE_LEVEL_NONE
  - CONFIG_APP_TRACE_LEVEL_ERROR
  - CONFIG_APP_TRACE_LEVEL_WARNING
  - CONFIG_APP_TRACE_LEVEL_API
  - CONFIG_APP_TRACE_LEVEL_EVENT
  - CONFIG_APP_TRACE_LEVEL_DEBUG
  - CONFIG_APP_TRACE_LEVEL_VERBOSE
- CONFIG_APP Anti-ROLLBACK (*CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*)
- CONFIG_APP_ROLLBACK_ENABLE (*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*)
- CONFIG_APP_SECURE_VERSION (*CONFIG_BOOTLOADER_APP_SECURE_VERSION*)
- CONFIG_APP_SECURE_VERSION_SIZE_EFUSE_FIELD (*CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD*)
- CONFIG_AVCIT_INITIAL_TRACE_LEVEL (*CONFIG_BT_LOG_AVCIT_TRACE_LEVEL*)
  - CONFIG_AVCIT_TRACE_LEVEL_NONE
  - CONFIG_AVCIT_TRACE_LEVEL_ERROR
  - CONFIG_AVCIT_TRACE_LEVEL_WARNING
  - CONFIG_AVCIT_TRACE_LEVEL_API
- CONFIG_AVCT_TRACE_LEVEL_EVENT
- CONFIG_AVCT_TRACE_LEVEL_DEBUG
- CONFIG_AVCT_TRACE_LEVEL_VERBOSE

**CONFIG_AVDT_INITIAL_TRACE_LEVEL** *(CONFIG_BT_LOG_AVDT_TRACE_LEVEL)*
- CONFIG_AVDT_TRACE_LEVEL_NONE
- CONFIG_AVDT_TRACE_LEVEL_ERROR
- CONFIG_AVDT_TRACE_LEVEL_WARNING
- CONFIG_AVDT_TRACE_LEVEL_API
- CONFIG_AVDT_TRACE_LEVEL_EVENT
- CONFIG_AVDT_TRACE_LEVEL_DEBUG
- CONFIG_AVDT_TRACE_LEVEL_VERBOSE

**CONFIG_AVRC_INITIAL_TRACE_LEVEL** *(CONFIG_BT_LOG_AVRC_TRACE_LEVEL)*
- CONFIG_AVRC_TRACE_LEVEL_NONE
- CONFIG_AVRC_TRACE_LEVEL_ERROR
- CONFIG_AVRC_TRACE_LEVEL_WARNING
- CONFIG_AVRC_TRACE_LEVEL_API
- CONFIG_AVRC_TRACE_LEVEL_EVENT
- CONFIG_AVRC_TRACE_LEVEL_DEBUG
- CONFIG_AVRC_TRACE_LEVEL_VERBOSE

**CONFIG_BLE_ACTIVE_SCAN_REPORT_ADV_SCAN_RSP_INDIVIDUALLY** *(CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN)*
- CONFIG_BLE_ADV_REPORT_DISCARD_THRESHOLD *(CONFIG_BTDM_BLE_ADV_REPORT_DISCARD_THRESHOLD)*
- CONFIG_BLE_ADV_REPORT_FLOW_CONTROL_NUM *(CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM)*
- CONFIG_BLE_ADV_REPORT_FLOW_CONTROL_SUPPORTED *(CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP)*
- CONFIG_BLE_ESTABLISH_LINK_CONNECTION_TIMEOUT *(CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT)*
- CONFIG_BLE_HOST_QUEUE_CONGESTION_CHECK *(CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK)*
- CONFIG_BLE_MESH_GATT_PROXY *(CONFIG_BT_BLE_MESH_GATT_PROXY_PASSEMBLY)*
- CONFIG_BLE_MESH_SCAN_DUPLICATE_EN *(CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN)*
- CONFIG_BLE_SCAN_DUPLICATE *(CONFIG_BTDM_BLE_SCAN_DUPL)*
- CONFIG_BLE_SMP_ENABLE *(CONFIG_BT_BLE_SMP_ENABLE)*
- CONFIG_BLUEDROID_MEM_DEBUG *(CONFIG_BT_BLUEDROID_MEM_DEBUG)*
- CONFIG_BLUEDROID_PINNED_TO_CORE_CHOICE *(CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE)*
- CONFIG_BLUEDROID_PINNED_TO_CORE_0
- CONFIG_BLUEDROID_PINNED_TO_CORE_1

**CONFIG_BLUFI_INITIAL_TRACE_LEVEL** *(CONFIG_BT_LOG_BLUFI_TRACE_LEVEL)*
- CONFIG_BLUFI_TRACE_LEVEL_NONE
- CONFIG_BLUFI_TRACE_LEVEL_ERROR
- CONFIG_BLUFI_TRACE_LEVEL_WARNING
- CONFIG_BLUFI_TRACE_LEVEL_API
- CONFIG_BLUFI_TRACE_LEVEL_EVENT
- CONFIG_BLUFI_TRACE_LEVEL_DEBUG
- CONFIG_BLUFI_TRACE_LEVEL_VERBOSE

**CONFIG_BNEP_INITIAL_TRACE_LEVEL** *(CONFIG_BT_LOG_BNEP_TRACE_LEVEL)*
- CONFIG_BROWNOUT_DET *(CONFIG_ESP_BROWNOUT_DET)*
- CONFIG_BROWNOUT_DET_LVL_SEL *(CONFIG_ESP_BROWNOUT_DET_LVL_SEL)*
- CONFIG_BTC_INITIAL_TRACE_LEVEL *(CONFIG_BT_LOG_BTC_TRACE_LEVEL)*
- CONFIG_BTC_TRACE_LEVEL_NONE

Submit Document Feedback
Chapter 2: API

- CONFIG_BTC_TRACE_LEVEL_ERROR
- CONFIG_BTC_TRACE_LEVEL_WARNING
- CONFIG_BTC_TRACE_LEVEL_API
- CONFIG_BTC_TRACE_LEVEL_EVENT
- CONFIG_BTC_TRACE_LEVEL_DEBUG
- CONFIG_BTC_TRACE_LEVEL_VERBOSE

- CONFIG_BTC_TASK_STACK_SIZE (CONFIG_BT_TASK_STACK_SIZE)
- CONFIG_BTDM_CONTROLLER_BLE_MAX_CONN (CONFIG_BTDM_CTRL_BLE_MAX_CONN)
- CONFIG_BTDM_CONTROLLER_BR_EDR_MAX_ACL_CONN (CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN)
- CONFIG_BTDM_CONTROLLER_BR_EDR_MAX_SYNC_CONN (CONFIG_BTDM_CTRL_BR_EDR_MAX_SYNC_CONN)
- CONFIG_BTDM_CONTROLLER_FULL_SCAN_SUPPORTED (CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED)
- CONFIG_BTDM_CONTROLLER_HCI_MODE_CHOICE (CONFIG_BTDM_CTRL_HCI_MODE_CHOICE)
  - CONFIG_BTDM_CONTROLLER_HCI_MODE_VHCI
  - CONFIG_BTDM_CONTROLLER_HCI_MODE_UART_H4
- CONFIG_BTDM_CONTROLLER_MODE (CONFIG_BTDM_CTRL_MODE)
  - CONFIG_BTDM_CONTROLLER_MODE_BLE_ONLY
  - CONFIG_BTDM_CONTROLLER_MODE_BR_EDR_ONLY
  - CONFIG_BTDM_CONTROLLER_MODE_BTDM
- CONFIG_BTDM_CONTROLLER_MODEM_SLEEP (CONFIG_BTDM_CTRL_MODEM_SLEEP)
- CONFIG_BTDM_CONTROLLER_PINNED_TO_CORE_CHOICE (CONFIG_BTDM_CTRL_PINNED_TO_CORE_CHOICE)
- CONFIG_BTIF_INITIAL_TRACE_LEVEL (CONFIG_BTIF_TRACE_LEVEL)
  - CONFIG_BTIF_TRACE_LEVEL_NONE
  - CONFIG_BTIF_TRACE_LEVEL_ERROR
  - CONFIG_BTIF_TRACE_LEVEL_WARNING
  - CONFIG_BTIF_TRACE_LEVEL_API
  - CONFIG_BTIF_TRACE_LEVEL_EVENT
  - CONFIG_BTIF_TRACE_LEVEL_DEBUG
  - CONFIG_BTIF_TRACE_LEVEL_VERBOSE
- CONFIG_BTM_INITIAL_TRACE_LEVEL (CONFIG_BTM_TRACE_LEVEL)
  - CONFIG_BTM_TRACE_LEVEL_NONE
  - CONFIG_BTM_TRACE_LEVEL_ERROR
  - CONFIG_BTM_TRACE_LEVEL_WARNING
  - CONFIG_BTM_TRACE_LEVEL_API
  - CONFIG_BTM_TRACE_LEVEL_EVENT
  - CONFIG_BTM_TRACE_LEVEL_DEBUG
  - CONFIG_BTM_TRACE_LEVEL_VERBOSE
- CONFIG_BTMU_TASK_STACK_SIZE (CONFIG_BTMU_TASK_STACK_SIZE)
- CONFIG_BT_NIMBLE_MSYS1_BLOCK_COUNT (CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT)
- CONFIG_BT_NIMBLE_TASK_STACK_SIZE (CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE)
- CONFIG_CLASSIC_BT_ENABLED (CONFIG_BT_CLASSIC_ENABLED)
- CONFIG_CONSOLE_UART (CONFIG_ESP_CONSOLE_UART)
  - CONFIG_CONSOLE_UART_DEFAULT
  - CONFIG_CONSOLE_UART_CUSTOM
  - CONFIG_CONSOLE_UART_NONE, CONFIG_ESP_CONSOLE_UART_NONE
- CONFIG_CONSOLE_UART_BAUDRATE (CONFIG_ESP_CONSOLE_UART_BAUDRATE)
• **CONFIG_CONSOLE_UART_NUM** *(CONFIG_ESP_CONSOLE_UART_NUM)*
  - CONFIG_CONSOLE_UART_CUSTOM_NUM_0
  - CONFIG_CONSOLE_UART_CUSTOM_NUM_1
• **CONFIG_CONSOLE_UART_RX_GPIO** *(CONFIG_ESP_CONSOLE_UART_RX_GPIO)*
• **CONFIG_CONSOLE_UART_TX_GPIO** *(CONFIG_ESP_CONSOLE_UART_TX_GPIO)*
• **CONFIG_CXX_EXCEPTIONS** *(CONFIG_COMPILER_CXX_EXCEPTIONS)*
• **CONFIG_CXX_EXCEPTIONS_EMG_POOL_SIZE** *(CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE)*
• **CONFIG_DUPLICATE_SCAN_CACHE_SIZE** *(CONFIG_BTDM_SCAN_DUPL_CACHE_SIZE)*
• **CONFIG_EFUSE_SECURE_VERSION_EMULATE** *(CONFIG_BOOTLOADER_EFUSE_SECURE_VERSION_EMULATE)*
• **CONFIG_ENABLE_STATIC_TASK_CLEAN_UP_HOOK** *(CONFIG_FREERTOS_ENABLE_STATIC_TASK_CLEAN_UP)*
• **CONFIG_ESP32_APTRACE_ONPANIC_HOST_FLUSH_TMO** *(CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO)*
• **CONFIG_ESP32_APTRACE_PENDING_DATA_SIZE_MAX** *(CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX)*
• **CONFIG_ESP32_APTRACE_POSTMORTEM_FLUSH_TRAX_THRESH** *(CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH)*
• **CONFIG_ESP32_COMPATIBLE_PRE_V2_1_BOOTLOADERS** *(CONFIG_APP_COMPATIBLE_PRE_V2_1_BOOTLOADERS)*
• **CONFIG_ESP32_COMPATIBLE_PRE_V3_1_BOOTLOADERS** *(CONFIG_APP_COMPATIBLE_PRE_V3_1_BOOTLOADERS)*
• **CONFIG_ESP32_CORE_DUMP_DECODE** *(CONFIG_ESP_COREDUMP_DECODE)*
  - CONFIG_ESP32_CORE_DUMP_DECODE_INFO
  - CONFIG_ESP32_CORE_DUMP_DECODE_DISABLE
• **CONFIG_ESP32_CORE_DUMP_MAX_TASKS_NUM** *(CONFIG_ESP_COREDUMP_MAX_TASKS_NUM)*
• **CONFIG_ESP32_CORE_DUMP_STACK_SIZE** *(CONFIG_ESP_COREDUMP_STACK_SIZE)*
• **CONFIG_ESP32_DEBUG_STUBS_ENABLE** *(CONFIG_ESP_DEBUG_STUBS_ENABLE)*
• **CONFIG_ESP32_GCOV_ENABLE** *(CONFIG_APPTRACE_GCOV_ENABLE)*
• **CONFIG_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE** *(CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE)*
• **CONFIG_ESP32_PHY_DEFAULT_INIT_IF_INVALID** *(CONFIG_ESP_PHY_DEFAULT_INIT_IF_INVALID)*
• **CONFIG_ESP32_PHY_INIT_DATA_ERROR** *(CONFIG_ESP_PHY_INIT_DATA_ERROR)*
• **CONFIG_ESP32_PHY_INIT_DATA_IN_PARTITION** *(CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION)*
• **CONFIG_ESP32_PHY_MAX_WIFI_TX_POWER** *(CONFIG_ESP_PHY_MAX_WIFI_TX_POWER)*
• **CONFIG_ESP32_PTHREAD_STACK_MIN** *(CONFIG_PTHREAD_STACK_MIN)*
• **CONFIG_ESP32_PTHREAD_TASK_CORE_DEFAULT** *(CONFIG_PTHREAD_TASK_CORE_DEFAULT)*
  - CONFIG_ESP32_DEFAULT_PTHREAD_CORE_NO_AFFINITY
  - CONFIG_ESP32_DEFAULT_PTHREAD_CORE_0
  - CONFIG_ESP32_DEFAULT_PTHREAD_CORE_1
• **CONFIG_ESP32_PTHREAD_TASK_NAME_DEFAULT** *(CONFIG_PTHREAD_TASK_NAME_DEFAULT)*
• **CONFIG_ESP32_PTHREAD_TASK_PRIO_DEFAULT** *(CONFIG_PTHREAD_TASK_PRIO_DEFAULT)*
• **CONFIG_ESP32_PTHREAD_TASK_STACK_SIZE_DEFAULT** *(CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT)*
• **CONFIG_ESP32_RTC_XTAL_BOOTSTRAP_CYCLES** *(CONFIG_ESP_SYSTEM_RTC_EXT_XTAL_BOOTSTRAP_CYCLES)*
• **CONFIG_ESP32_SUPPORT_MULTIPLE_PHY_INIT_DATA_BIN** *(CONFIG_ESP_PHY_MULTIPLE_INIT_DATA_BIN)*
• **CONFIG_ESP_GRAUHTIOUS_ARP** *(CONFIG_LWIP_ESP_GRATUITOUS_ARP)*
• **CONFIG_ESP_SYSTEM_PD_FLASH** *(CONFIG_ESP_SLEEP_POWER_DOWN_FLASH)*
• **CONFIG_ESP_TASK_WDT** *(CONFIG_ESP_TASK_WDT_INIT)*
• **CONFIG_EVENT_LOOP_PROFILING** *(CONFIG_ESP_EVENT_LOOP_PROFILING)*
• **CONFIG_FLASH_ENCRYPTION_ENABLED** *(CONFIG_SECURE_FLASH_ENC_ENABLED)*
• **CONFIG_FLASH_ENCRYPTION_UART_BOOTLOADER_ALLOW_CACHE** *(CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE)*
• **CONFIG_GAP_INITIAL_TRACE_LEVEL** *(CONFIG_BT_LOG_GAP_TRACE_LEVEL)*
– CONFIG_GAP_TRACE_LEVEL_NONE
– CONFIG_GAP_TRACE_LEVEL_ERROR
– CONFIG_GAP_TRACE_LEVEL_WARNING
– CONFIG_GAP_TRACE_LEVEL_API
– CONFIG_GAP_TRACE_LEVEL_EVENT
– CONFIG_GAP_TRACE_LEVEL_DEBUG
– CONFIG_GAP_TRACE_LEVEL_VERBOSE

• CONFIG_GAP_TMR_INTERVAL (CONFIG_LWIP_GAP_TMR_INTERVAL)
• CONFIG_GATTC_CACHE_NV_FLASH (CONFIG_BT_GATTC_CACHE_NV_FLASH)
• CONFIG_GATTC_ENABLE (CONFIG_BT_GATTC_ENABLE)
• CONFIG_GATTS_ENABLE (CONFIG_BT_GATTS_ENABLE)
• CONFIG_GATTS_SEND_SERVICE_CHANGE_MODE (CONFIG_BT_GATTS_SEND_SERVICE_CHANGE_MODE)

– CONFIG_GATTS_SEND_SERVICE_CHANGE_MANUAL
– CONFIG_GATTS_SEND_SERVICE_CHANGE_AUTO

• CONFIG_GATT_INITIAL_TRACE_LEVEL (CONFIG_BT_LOG_GATT_TRACE_LEVEL)
  – CONFIG_GATT_TRACE_LEVEL_NONE
  – CONFIG_GATT_TRACE_LEVEL_ERROR
  – CONFIG_GATT_TRACE_LEVEL_WARNING
  – CONFIG_GATT_TRACE_LEVEL_API
  – CONFIG_GATT_TRACE_LEVEL_EVENT
  – CONFIG_GATT_TRACE_LEVEL_DEBUG
  – CONFIG_GATT_TRACE_LEVEL_VERBOSE

• CONFIG_GDBSTUB_MAX_TASKS (CONFIG_ESP_GDBSTUB_MAX_TASKS)
• CONFIG_GDBSTUB_SUPPORT_TASKS (CONFIG_ESP_GDBSTUB_SUPPORT_TASKS)

• CONFIG_HCI_INITIAL_TRACE_LEVEL (CONFIG_BT_LOG_HCI_TRACE_LEVEL)
  – CONFIG_HCI_TRACE_LEVEL_NONE
  – CONFIG_HCI_TRACE_LEVEL_ERROR
  – CONFIG_HCI_TRACE_LEVEL_WARNING
  – CONFIG_HCI_TRACE_LEVEL_API
  – CONFIG_HCI_TRACE_LEVEL_EVENT
  – CONFIG_HCI_TRACE_LEVEL_DEBUG
  – CONFIG_HCI_TRACE_LEVEL_VERBOSE

• CONFIG_HFP_AUDIO_DATA_PATH (CONFIG_BT_HFP_AUDIO_DATA_PATH)
  – CONFIG_HFP_AUDIO_DATA_PATH_PCM
  – CONFIG_HFP_AUDIO_DATA_PATH_HCI

• CONFIG_HFP_ENABLE (CONFIG_BT_HFP_ENABLE)

• CONFIG_HFP_ROLE (CONFIG_BT_HFP_ROLE)
  – CONFIG_HFP_CLIENT_ENABLE
  – CONFIG_HFP_AG_ENABLE

• CONFIG_HID_INITIAL_TRACE_LEVEL (CONFIG_BT_LOG_HID_TRACE_LEVEL)
  – CONFIG_HID_TRACE_LEVEL_NONE
  – CONFIG_HID_TRACE_LEVEL_ERROR
  – CONFIG_HID_TRACE_LEVEL_WARNING
  – CONFIG_HID_TRACE_LEVEL_API
  – CONFIG_HID_TRACE_LEVEL_EVENT
  – CONFIG_HID_TRACE_LEVEL_DEBUG
  – CONFIG_HID_TRACE_LEVEL_VERBOSE

• CONFIG_INT_WDT (CONFIG_ESP_INT_WDT)
• CONFIG_INT_WDT_CHECK_CPU1 (CONFIG_ESP_INT_WDT_CHECK_CPU1)
• CONFIG_INT_WDT_TIMEOUT_MS (CONFIG_ESP_INT_WDT_TIMEOUT_MS)
• CONFIG_IPC_TASK_STACK_SIZE (CONFIG_ESP_IPC_TASK_STACK_SIZE)

• CONFIG_L2CAP_INITIAL_TRACE_LEVEL (CONFIG_BT_LOG_L2CAP_TRACE_LEVEL)
  – CONFIG_L2CAP_TRACE_LEVEL_NONE
  – CONFIG_L2CAP_TRACE_LEVEL_ERROR
  – CONFIG_L2CAP_TRACE_LEVEL_WARNING
  – CONFIG_L2CAP_TRACE_LEVEL_API
  – CONFIG_L2CAP_TRACE_LEVEL_EVENT
- CONFIG_L2CAP_TRACE_LEVEL_DEBUG
- CONFIG_L2CAP_TRACE_LEVEL_VERBOSE

- CONFIG_LOG_BOOTLOADER_LEVEL (CONFIG_BOOTLOADER_LOG_LEVEL)
  - CONFIG_LOG_BOOTLOADER_LEVEL_NONE
  - CONFIG_LOG_BOOTLOADER_LEVEL_ERROR
  - CONFIG_LOG_BOOTLOADER_LEVEL_WARN
  - CONFIG_LOG_BOOTLOADER_LEVEL_INFO
  - CONFIG_LOG_BOOTLOADER_LEVEL_DEBUG
  - CONFIG_LOG_BOOTLOADER_LEVEL_VERBOSE

- CONFIG_MAIN_TASK_STACK_SIZE (CONFIG_ESP_MAIN_TASK_STACK_SIZE)

- CONFIG_MCA_INITIALIZATION_TRACE_LEVEL (CONFIG_BT_LOG_MCA_TRACE_LEVEL)
  - CONFIG_MCA_TRACE_LEVEL_NONE
  - CONFIG_MCA_TRACE_LEVEL_ERROR
  - CONFIG_MCA_TRACE_LEVEL_WARNING
  - CONFIG_MCA_TRACE_LEVEL_API
  - CONFIG_MCA_TRACE_LEVEL_EVENT
  - CONFIG_MCA_TRACE_LEVEL_DEBUG
  - CONFIG_MCA_TRACE_LEVEL_VERBOSE

- CONFIG_MCPWM_ISR_IN_IRAM (CONFIG_MCPWM_ISR_IRAM_SAFE)

- CONFIG_MESH_DUPLICATE_SCAN_CACHE_SIZE (CONFIG_BTDM_MESH_DUPL_SCAN_CACHE_SIZE)

- CONFIG_NIMBLE_ACL_BUF_COUNT (CONFIG_BT_NIMBLE_ACL_BUF_COUNT)
- CONFIG_NIMBLE_ACL_BUF_SIZE (CONFIG_BT_NIMBLE_ACL_BUF_SIZE)

- CONFIG_NIMBLE_ATT_PREFERRED_MTU (CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU)

- CONFIG_NIMBLE_CRYPTO_STACK_MBEDTLS (CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS)

- CONFIG_NIMBLE_DEBUG (CONFIG_BT_NIMBLE_DEBUG)

- CONFIG_NIMBLE_GAP_DEVICE_NAME_MAX_LEN (CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN)

- CONFIG_NIMBLE_HCI_EVT_BUF_SIZE (CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE)

- CONFIG_NIMBLE_HCI_EVT_HI_BUF_COUNT (CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT)

- CONFIG_NIMBLE_HCI_EVT_LO_BUF_COUNT (CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT)

- CONFIG_NIMBLE_HS_FLOW_CTRL (CONFIG_BT_NIMBLE_HS_FLOW_CTRL)

- CONFIG_NIMBLE_HS_FLOW_CTRL_THR (CONFIG_BT_NIMBLE_HS_FLOW_CTRL_THR)

- CONFIG_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT (CONFIG_BT_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT)

- CONFIG_NIMBLE_L2CAP_COC_MAX_NUM (CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM)

- CONFIG_NIMBLE_MAX_BONDS (CONFIG_BT_NIMBLE_MAX_BONDS)

- CONFIG_NIMBLE_MAX_CCCDS (CONFIG_BT_NIMBLE_MAX_CCCDS)

- CONFIG_NIMBLE_MAX_CONNECTIONS (CONFIG_BT_NIMBLE_MAX_CONNECTIONS)

- CONFIG_NIMBLE_MEM_ALLOC_MODE (CONFIG_BT_NIMBLE_MEM_ALLOC_MODE)
  - CONFIG_NIMBLE_MEMALLOC_MODE_INTERNAL
  - CONFIG_NIMBLE_MEMALLOC_MODE_EXTERNAL
  - CONFIG_NIMBLE_MEMALLOC_MODE_DEFAULT

- CONFIG_NIMBLE_MESH (CONFIG_BT_NIMBLE_MESH)

- CONFIG_NIMBLE_MESH_DEVICE_NAME (CONFIG_BT_NIMBLE_MESHDEVICE_NAME)

- CONFIG_NIMBLE_MESH_FRIEND (CONFIG_BT_NIMBLE_MESH_FRIEND)

- CONFIG_NIMBLE_MESH_GATT_PROXY (CONFIG_BT_NIMBLE_MESH_GATT_PROXY)

- CONFIG_NIMBLE_MESH_LOW_POWER (CONFIG_BT_NIMBLE_MESH_LOW_POWER)

- CONFIG_NIMBLE_MESH_PB ADV (CONFIG_BT_NIMBLE_MESH_PB_ADV)

- CONFIG_NIMBLE_MESH_PB_GATT (CONFIG_BT_NIMBLE_MESH_PB_GATT)

- CONFIG_NIMBLE_MESH_PROXY (CONFIG_BT_NIMBLE_MESH_PROXY)

- CONFIG_NIMBLE_MESH_RELAY (CONFIG_BT_NIMBLE_MESH_RELAY)

- CONFIG_NIMBLE_NVSS_PERSIST (CONFIG_BT_NIMBLE_NVSS_PERSIST)

- CONFIG_NIMBLE_PINNED_TO_CORE_CHOICE (CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE)
  - CONFIG_NIMBLE_PINNED_TO_CORE_0
  - CONFIG_NIMBLE_PINNED_TO_CORE_1
• CONFIG_NIMBLE_ROLE_BROADCASTER
• CONFIG_NIMBLE_ROLE_CENTRAL
• CONFIG_NIMBLE_ROLE_OBSERVER
• CONFIG_NIMBLE_ROLE_PERIPHERAL
• CONFIG_NIMBLE_RPA_TIMEOUT
• CONFIG_NIMBLE_SM_LEGACY
• CONFIG_NIMBLE_SM_SC
• CONFIG_NIMBLE_SM_SC_DEBUG_KEYS
• CONFIG_NIMBLE_SVC_GAP_APPEARANCE
• CONFIG_NIMBLE_SVC_GAP_DEVICE_NAME
• CONFIG_NIMBLE_TASK_STACK_SIZE
• CONFIG_NO_BLOBS
• CONFIG_NUMBER_OF_UNIVERSAL_MAC_ADDRESS
• CONFIG_OPTIMIZATION_ASSERTION_LEVEL
• CONFIG_OPTIMIZATION_COMPILER
• CONFIG_OSI_INITIAL_TRACE_LEVEL
• CONFIG_PAN_INITIAL_TRACE_LEVEL
• CONFIG_POST_EVENTS_FROM_IRAM_ISR
• CONFIG_POST_EVENTS_FROM_ISR
• CONFIG_PPP_CHAP_SUPPORT
• CONFIG_PPP_DEBUG_ON
• CONFIG_PPP_MPPE_SUPPORT
• CONFIG_PPP_MSCAP_SUPPORT
• CONFIG_PPP_NOTIFY_PHASE_SUPPORT
• CONFIG_PPP_PAP_SUPPORT
• CONFIG_PPP_SUPPORT
• CONFIG_REDUCE_PHY_TX_POWER
• CONFIG_RFCOMM_INITIAL_TRACE_LEVEL
• CONFIG_POST_EVENTS_FROM_IRAM_ISR
• CONFIG_POST_EVENTS_FROM_ISR
• CONFIG_PPP_CHAP_SUPPORT
• CONFIG_PPP_DEBUG_ON
• CONFIG_PPP_MPPE_SUPPORT
• CONFIG_PPP_MSCAP_SUPPORT
• CONFIG_PPP_NOTIFY_PHASE_SUPPORT
• CONFIG_PPP_PAP_SUPPORT
• CONFIG_PPP_SUPPORT
• CONFIG_REDUCE_PHY_TX_POWER
• CONFIG_RFCOMM_INITIAL_TRACE_LEVEL
- CONFIG_RFCOMM_TRACE_LEVEL_VERBOSE

- **CONFIG_SCAN_DUPLICATE_TYPE** (CONFIG_BTDM_SCAN_DUPL_TYPE)
  - CONFIG_SCAN_DUPLICATE_BY_DEVICE_ADDR
  - CONFIG_SCAN_DUPLICATE_BY_ADV_DATA
  - CONFIG_SCAN_DUPLICATE_BY_ADV_DATA_AND_DEVICE_ADDR

- **CONFIG_SEMIHOSTFS_MAX_MOUNT_POINTS** (CONFIG_VFS_SEMIHOSTFS_MAX_MOUNT_POINTS)

- **CONFIG_SMP_INITIAL_TRACE_LEVEL** (CONFIG_BT_LOG_SMP_TRACE_LEVEL)
  - CONFIG_SMP_TRACE_LEVEL_NONE
  - CONFIG_SMP_TRACE_LEVEL_ERROR
  - CONFIG_SMP_TRACE_LEVEL_WARNING
  - CONFIG_SMP_TRACE_LEVEL_API
  - CONFIG_SMP_TRACE_LEVEL_EVENT
  - CONFIG_SMP_TRACE_LEVEL_DEBUG
  - CONFIG_SMP_TRACE_LEVEL_VERBOSE

- **CONFIG_SMP_SLAVE_CON_PARAMS_UPD_ENABLE** (CONFIG_BT_SMP_SLAVE_CON_PARAMS_UPD_ENABLE)

- **CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS** (CONFIG_SPI_FLASH_DANGEROUS_WRITE)
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_ABORTS
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_FAILS
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_ALLOWED

- **CONFIG_STACK_CHECK_MODE** (CONFIG_COMPILER_STACK_CHECK_MODE)
  - CONFIG_STACK_CHECK_NONE
  - CONFIG_STACK_CHECK_NORM
  - CONFIG_STACK_CHECK_STRONG
  - CONFIG_STACK_CHECK_ALL

- **CONFIG_SUPPORT_TERMIOS** (CONFIG_VFS_SUPPORT_TERMIOS)

- **CONFIG_SUPPRESS_SELECT_DEBUG_OUTPUT** (CONFIG_VFS_SUPPRESS_SELECT_DEBUG_OUTPUT)

- **CONFIG_SYSTEM_EVENT_QUEUE_SIZE** (CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE)

- **CONFIG_SYSVIEW_BUF_WAIT_TMO** (CONFIG_APPTRACE_SV_BUF_WAIT_TMO)

- **CONFIG_SYSVIEW_ENABLE** (CONFIG_APPTRACE_SV_ENABLE)

- **CONFIG_SYSVIEW_EVT_IDLE_ENABLE** (CONFIG_APPTRACE_SV_EVT_IDLE_ENABLE)

- **CONFIG_SYSVIEW_EVT_ISR_ENTER_ENABLE** (CONFIG_APPTRACE_SV_EVT_ISR_ENTER_ENABLE)

- **CONFIG_SYSVIEW_EVT_ISR_EXIT_ENABLE** (CONFIG_APPTRACE_SV_EVT_ISR_EXIT_ENABLE)

- **CONFIG_SYSVIEW_EVT_ISR_TO_SCHEDULER_ENABLE** (CONFIG_APPTRACE_SV_EVT_ISR_TO_SCHED_ENABLE)

- **CONFIG_SYSVIEW_EVT_OVERFLOW_ENABLE** (CONFIG_APPTRACE_SV_EVT_OVERFLOW_ENABLE)

- **CONFIG_SYSVIEW_EVT_TASK_CREATE_ENABLE** (CONFIG_APPTRACE_SV_EVT_TASK_CREATE_ENABLE)

- **CONFIG_SYSVIEW_EVT_TASK_INTERRUPT_ENABLE** (CONFIG_APPTRACE_SV_EVT_TASK_INTERRUPT_ENABLE)

- **CONFIG_SYSVIEW_EVT_TASK_STOP_EXEC_ENABLE** (CONFIG_APPTRACE_SV_EVT_TASK_STOP_EXEC_ENABLE)

- **CONFIG_SYSVIEW_EVT_TASK_STOP_READY_ENABLE** (CONFIG_APPTRACE_SV_EVT_TASK_STOP_READY_ENABLE)

- **CONFIG_SYSVIEW_EVT_TIMER_ENTER_ENABLE** (CONFIG_APPTRACE_SV_EVT_TIMER_ENTER_ENABLE)

- **CONFIG_SYSVIEW_EVT_TIMER_EXIT_ENABLE** (CONFIG_APPTRACE_SV_EVT_TIMER_EXIT_ENABLE)

- **CONFIG_SYSVIEW_MAX_TASKS** (CONFIG_APPTRACE_SV_MAX_TASKS)

- **CONFIG_SYSVIEW_TS_SOURCE** (CONFIG_APPTRACE_SV_TS_SOURCE)
  - CONFIG_SYSVIEW_TS_SOURCE_CCOUNT
  - CONFIG_SYSVIEW_TS_SOURCE_ESP_TIMER

- **CONFIG_TASK_WDT_CHECK_IDLE_TASK_CPU0** (CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0)

- **CONFIG_TASK_WDT_CHECK_IDLE_TASK_CPU1** (CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1)

- **CONFIG_TASK_WDT_PANIC** (CONFIG_ESP_TASK_WDT_PANIC)

- **CONFIG_TASK_WDT_TIMEOUT_S** (CONFIG_ESP_TASK_WDT_TIMEOUT_S)

- **CONFIG_TCPIP_RECVMBOX_SIZE** (CONFIG_LWIP_TCPIP_RECVMBOX_SIZE)

- **CONFIG_TCPIP_TASK_AFFINITY** (CONFIG_LWIP_TCPIP_TASK_AFFINITY)
  - CONFIG_TCPIP_TASK_AFFINITY_NO_AFFINITY
  - CONFIG_TCPIP_TASK_AFFINITY_CPU0
  - CONFIG_TCPIP_TASK_AFFINITY_CPU1
2.8 配网 API

2.8.1 Protocol Communication

Overview

Protocol Communication (protocomm) component manages secure sessions and provides framework for multiple transports. The application can also use protocomm layer directly to have application specific extensions for the provisioning (or non-provisioning) use cases.

Following features are available for provisioning :

- Communication security at application level -
  - protocomm_security0 (no security)
  - protocomm_security1 (Curve25519 key exchange + AES-CTR encryption/decryption)
  - protocomm_security2 (SRP6a-based key exchange + AES-GCM encryption/decryption)

- Proof-of-possession (support with protocomm_security1 only)
- Salt and Verifier (support with protocomm_security2 only)

Protocomm internally uses protobuf (protocol buffers) for secure session establishment. Though users can implement their own security (even without using protobuf). One can even use protocomm without any security layer.

Protocomm provides framework for various transports - WiFi (SoftAP+HTTPD), BLE, console - in which case the handler invocation is automatically taken care of on the device side (see Transport Examples below for code snippets).

Note that the client still needs to establish session (for protocomm_security1 and protocomm_security2) by performing the two way handshake. See Unified Provisioning for more details about the secure handshake logic.

Enabling protocomm security version

Protocomm component provides project configuration menu to enable/disable support of respective security versions. The respective configuration options can be found as follows:

- Support protocomm security version 0 (no security): \texttt{CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0}
  (this option is enabled by default)
Chapter 2. API 参考

- Support protocomm security version 1 (Curve25519 key exchange + AES-CTR encryption/decryption): `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1` (this option is enabled by default)
- Support protocomm security version 2 (SRP6a-based key exchange + AES-GCM encryption/decryption): `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2`

备注：Enabling multiple security versions allow to control them dynamically but also increases firmware size.

Transport Example (SoftAP + HTTP) with Security 2

For sample usage, see [wifi_provisioning/src/scheme_softap.c](https://github.com/espressif/esp-idf/blob/master/wifi_provisioning/src/scheme_softap.c)

```c
static const char sec2_salt[] = {0x7f, 0x5f, 0xe2, 0xbe, 0xba, 0xc7, 0x81, 0x0c};
static const char sec2_verifier[] = {0x83, 0x4b, inlen, 0xf7, 0xef, outlen, 0x23, 0x11, 0xac, 0x23, 0x14, 0x7c, 0x2a, 0xe5, 0x5c, 0x07, 0x7a, 0x93, 0x8a};

* This simply echoes back the received data. *
static esp_err_t echo_req_handler (uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)
{
    /* Session ID may be used for persistence */
    printf("Session ID : %d", session_id);

    /* Echo back the received data */
    *outlen = inlen;
    /* Output data length updated */
    *outbuf = malloc(inlen);
    /* This will be deallocated outside */
    memcpy(*outbuf, inbuf, inlen);

    /* Private data that was passed at the time of endpoint creation */
    if (priv) {
        printf("Private data : %d", *priv);
    }

    return ESP_OK;
}
```

（下頁繼續）
Chapter 2. API 參考

(續上頁)

```c
0xfb, 0x52, 0x22, 0x3b, 0x7a, 0x7b, 0x9e, 0xe9, 0xee, 0xc, 0x44,_
-0xda0, 0x73, 0x72, 0x2a, 0xca, 0x85, 0x15, 0x4a,
-0x60, 0xce, 0xa4, 0x8, 0x7d, 0x57, 0xa4, 0xf8, 0x77, 0x22, 0xc1,_
-0xa5, 0xfa, 0xfb, 0x71, 0x91, 0x3b, 0xfe, 0x87,
-0x51, 0xfe, 0x05, 0x2d, 0xd6, 0xd3, 0x74, 0xe5, 0xe2, 0x68, 0x79,_
-0x34, 0x70, 0x40, 0x12, 0xa8, 0xe1, 0xb4, 0x8c, 0x4c,
-0x8a, 0x46, 0x73, 0xc, 0x8d, 0x72, 0x67, 0x32, 0x42, 0xdc,_
-0x10, 0x33, 0x71, 0xe8, 0x8b, 0x00, 0xe6, 0x29b,
-0x0a, 0xe9, 0xb4, 0x8f, 0xe8b, 0x70, 0x52, 0x8d, 0x0a, 0xc1, 0x7e,_
-0x2e, 0xb0, 0x61, 0xa6, 0xe1, 0xa3, 0x34, 0x4b,
-0x2a, 0x3c, 0xc4, 0x5d, 0x42, 0x05, 0x58, 0x25, 0x3d, 0xca, 0x96,_
-0x5c, 0xb9, 0x52, 0xf9, 0xe8, 0x80, 0x75, 0x3d,
-0xc8, 0x9f, 0x7, 0xb2, 0xaa, 0x95, 0xe2, 0x76, 0xb3, 0xe1, 0x48,_
-0xc1, 0xa0, 0xa1, 0x9a, 0xe8, 0xf, 0x41, 0x28,
-0xd2, 0x16, 0xe1, 0xa6, 0x8d, 0x73, 0x51, 0x73, 0x79, 0x98, 0xd9,_
-0xb9, 0x00, 0x50, 0xa2, 0x4d, 0x99, 0x18, 0x90,
-0x70, 0x27, 0xe7, 0x8d, 0x56, 0x45, 0x34, 0xe1, 0xb9, 0x30, 0xda,_
-0xe, 0x4a, 0x08, 0x27, 0x9f, 0xfa, 0x59, 0xe2e,
-0x36, 0x77, 0x00, 0xe2, 0xb6, 0xe8b, 0xd1, 0x56, 0x50, 0x8e};

/* Example function for launching a protocomm instance over HTTP */
protocomm_t *start_pc()
{
    protocomm_t *pc = protocomm_new();

    /* Config for protocomm_httpd_start() */
    protocomm_httpd_config_t pc_config = {
        .data = {
            .config = PROTOCOL_HTTPD_DEFAULT_CONFIG()
        }
    };

    /* Start protocomm server on top of HTTP */
    protocomm_httpd_start(pc, &pc_config);

    /* Create Security2 params object from salt and verifier. It must be_
     * valid
    * throughout the scope of protocomm endpoint. This need not be_
    * static,
     * i.e. could be dynamically allocated and freed at the time of_
    * end point_
     * removal */
    const static protocomm_security2_params_t sec2_params = {
        .salt = (const uint8_t *) salt,
        .salt_len = sizeof(salt),
        .verifier = (const uint8_t *) verifier,
        .verifier_len = sizeof(verifier),
    };

    /* Set security for communication at application level. Just like for_
     * request handlers, setting security creates an end point and_
     * registers
     * the handler provided by protocomm_security1. One can similarly use_
     * protocomm_security0. Only one type of security can be set for a_
     * protocomm instance at a time. */
    protocomm_set_security(pc, "security_endpoint", &protocomm_security2,_
                           sec2_params);

    /* Private data passed to the end point must be valid throughout the_
    *(下頁續)*/
```
Chapter 2. API

Transport Example (SoftAP + HTTP) with Security 1

For sample usage, see wifi_provisioning/src/scheme_softap.c

```
/* Endpoint handler to be registered with protocomm. */
* This simply echoes back the received data. */
esp_err_t echo_req_handler (uint32_t session_id,
   const uint8_t *inbuf, ssize_t inlen,
   uint8_t **outbuf, ssize_t *outlen,
   void *priv_data)
{

   /* Session ID may be used for persistence */
   printf("Session ID : %d", session_id);

   /* Echo back the received data */
   *outlen = inlen;  /* Output data length updated */
   *outbuf = malloc(inlen);  /* This will be deallocated outside */
   memcpy(*outbuf, inbuf, inlen);

   /* Private data that was passed at the time of endpoint creation */
   uint32_t *priv = (uint32_t *) priv_data;
   if (priv) {
       printf("Private data : %d", *priv);
   }

   return ESP_OK;
}
```

** of protocomm endpoint. This need not be static, ie. could be
* dynamically
* allocated and freed at the time of endpoint removal */
static uint32_t priv_data = 1234;
/* Add a new endpoint for the protocomm instance, identified by a_
* unique name
* and register a handler function along with private data to be_
* passed at the_
* time of handler execution. Multiple endpoints can be added as long_
* as they
* are identified by unique names */
protocomm_add_endpoint(pc, "echo_req_endpoint",
   echo_req_handler, (void *) &priv_data);

```
/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc)
{
   /* Remove endpoint identified by it's unique name */
   protocomm_remove_endpoint(pc, "echo_req_endpoint");

   /* Remove security endpoint identified by it's name */
   protocomm_unset_security(pc, "security_endpoint");

   /* Stop HTTP server */
   protocomm_httpd_stop(pc);

   /* Delete (deallocate) the protocomm instance */
   protocomm_delete(pc);
}
```

Submit Document Feedback
Release v5.1-dev-2066-g7869f4e151
/* Example function for launching a protocomm instance over HTTP */
protocomm_t *start_pc(const char *pop_string)
{
    protocomm_t *pc = protocomm_new();

    /* Config for protocomm_httpd_start() */
    protocomm_httpd_config_t pc_config = {
        .data = {
            .config = PROTOCOL_HTTPD_DEFAULT_CONFIG()
        }
    };

    /* Start protocomm server on top of HTTP */
    protocomm_httpd_start(pc, &pc_config);

    /* Create security1 params object from pop_string. It must be valid */
    /* through the scope of protocomm endpoint. This need not be_*/
    /* static, */
    /* i.e. could be dynamically allocated and freed at the time of_*/
    /* endpoint */
    /* removal */
    const static protocomm_security1_params_t sec1_params = {
        .data = (const uint8_t *) strdup(pop_string),
        .len = strlen(pop_string)
    };

    /* Set security for communication at application level. Just like for */
    /* request handlers, setting security creates an endpoint and_*/
    /* registers */
    /* the handler provided by protocomm_security1. One can similarly use */
    /* protocomm_security0. Only one type of security can be set for a */
    /* protocomm instance at a time. */
    protocomm_set_security(pc, "security_endpoint", &protocomm_security1, /*sec1_params*/);

    /* Private data passed to the endpoint must be valid throughout the_*/
    /* scope */
    /* of protocomm endpoint. This need not be static, i.e. could be_*/
    /* dynamically */
    /* allocated and freed at the time of endpoint removal */
    static uint32_t priv_data = 1234;

    /* Add a new endpoint for the protocomm instance, identified by a_*/
    /* unique name */
    /* and register a handler function along with private data to be_*/
    /* passed at the */
    /* time of handler execution. Multiple endpoints can be added as long_*/
    /* as they */
    /* are identified by unique names */
    protocomm_add_endpoint(pc, "echo_req_endpoint",
                           echo_req_handler, (void *) &priv_data);
    return pc;
}

/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc)
{
    /* Remove endpoint identified by it's unique name */
    protocomm_remove_endpoint(pc, "echo_req_endpoint");
}
/* Remove security endpoint identified by it's name */
protocomm_unset_security(pc, "security_endpoint");

/* Stop HTTP server */
protocomm_httpd_stop(pc);

/* Delete (deallocate) the protocomm instance */
protocomm_delete(pc);

Transport Example (BLE) with Security 0

For sample usage, see wifi_provisioning/src/scheme_ble.c

/* Example function for launching a secure protocomm instance over BLE */
protocomm_t *start_pc() {
    protocomm_t *pc = protocomm_new();

    /* Endpoint UUIDs */
    protocomm_ble_name_uuid_t nu_lookup_table[] = {
        {"security_endpoint", 0xFF51},
        {"echo_req_endpoint", 0xFF52}
    };

    /* Config for protocomm_ble_start() */
    protocomm_ble_config_t config = {
        .service_uuid = {
            /* LSB <---------------------------------------
             * ---------------------------------------> MSB */
            0xfb, 0x34, 0x9b, 0x80, 0x80, 0x00, 0x80,
            0x00, 0x10, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00,
        },
        .nu_lookup_count = sizeof(nu_lookup_table)/sizeof(nu_lookup_table[0]),
        .nu_lookup = nu_lookup_table
    };

    /* Start protocomm layer on top of BLE */
    protocomm_ble_start(pc, &config);

    /* For protocomm_security0, Proof of Possession is not used, and can be kept NULL */
    protocomm_set_security(pc, "security_endpoint", (protocomm_security0, _NULL),
        protocomm_add_endpoint(pc, "echo_req_endpoint", echo_req_handler, _NULL);,
        return pc;
}

/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc) {
    protocomm_remove_endpoint(pc, "echo_req_endpoint");
    protocomm_unset_security(pc, "security_endpoint");

    /* Stop BLE protocomm service */
    protocomm_ble_stop(pc);
Chapter 2. API 参考

API Reference

Header File

- components/protocomm/include/common/protocomm.h

Functions

`protocomm_t *protocomm_new(void)`
Create a new protocomm instance.

This API will return a new dynamically allocated protocomm instance with all elements of the `protocomm_t` structure initialized to NULL.

- `protocomm_t *`: On success
- `NULL`: No memory for allocating new instance

`void protocomm_delete(protocomm_t *pc)`
Delete a protocomm instance.

This API will deallocate a protocomm instance that was created using `protocomm_new()`.

- `pc` - [in] Pointer to the protocomm instance to be deleted

`esp_err_t protocomm_add_endpoint(protocomm_t *pc, const char *ep_name, protocomm_req_handler_t h, void *priv_data)`
Add endpoint request handler for a protocomm instance.

This API will bind an endpoint handler function to the specified endpoint name, along with any private data that needs to be pass to the handler at the time of call.

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- This function internally calls the registered `add_endpoint()` function of the selected transport which is a member of the `protocomm_t` instance structure.

- `pc` - [in] Pointer to the protocomm instance
- `ep_name` - [in] Endpoint identifier(name) string
- `h` - [in] Endpoint handler function
- `priv_data` - [in] Pointer to private data to be passed as a parameter to the handler function on call. Pass NULL if not needed.

- `ESP_OK`: Success
- `ESP_FAIL`: Error adding endpoint / Endpoint with this name already exists
- `ESP_ERR_NO_MEM`: Error allocating endpoint resource
- `ESP_ERR_INVALID_ARG`: Null instance/name/handler arguments

`esp_err_t protocomm_remove_endpoint(protocomm_t *pc, const char *ep_name)`
Remove endpoint request handler for a protocomm instance.

This API will remove a registered endpoint handler identified by an endpoint name.

-备注:

Espressif Systems

Submit Document Feedback
This function internally calls the registered `remove_endpoint()` function which is a member of the `protocomm_t` instance structure.

**Parameters**
- `pc` [in] Pointer to the protocomm instance
- `ep_name` [in] Endpoint identifier (name) string

**Return**
- `ESP_OK`: Success
- `ESP_ERR_NOT_FOUND`: Endpoint with specified name doesn’t exist
- `ESP_ERR_INVALID_ARG`: Null instance/name arguments

```c
esp_err_t protocomm_open_session(protocomm_t *pc, uint32_t session_id)
```

Allocates internal resources for new transport session.

**Notes:**
- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

```c
esp_err_t protocomm_close_session(protocomm_t *pc, uint32_t session_id)
```

Frees internal resources used by a transport session.

**Notes:**
- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

```c
esp_err_t protocomm_req_handle(protocomm_t *pc, const char *ep_name, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen)
```

Calls the registered handler of an endpoint session for processing incoming data and generating the response.

**Notes:**
- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- Resulting output buffer must be deallocated by the caller.
Chapter 2. API

• **inbuf**: [in] Input buffer contains input request data which is to be processed by the registered handler
• **inlen**: [in] Length of the input buffer
• **outbuf**: [out] Pointer to internally allocated output buffer, where the resulting response data output from the registered handler is to be stored
• **outlen**: [out] Buffer length of the allocated output buffer

ESP_OK: Request handled successfully
ESP_FAIL: Internal error in execution of registered handler
ESP_ERR_NO_MEM: Error allocating internal resource
ESP_ERR_NOT_FOUND: Endpoint with specified name doesn’t exist
ESP_ERR_INVALID_ARG: Null instance/name arguments

```c
esp_err_t protocomm_set_security(protocomm_t *pc, const char* ep_name, const protocomm_security_t *sec, const void *sec_params)
```

Add endpoint security for a protocomm instance.

This API will bind a security session establisher to the specified endpoint name, along with any proof of possession that may be required for authenticating a session client.

**备注:**

• An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
• The choice of security can be any `protocomm_security_t` instance. Choices `protocomm_security0` and `protocomm_security1` and `protocomm_security2` are readily available.

```c
esp_err_t protocomm_unset_security(protocomm_t *pc, const char* ep_name)
```

Remove endpoint security for a protocomm instance.

This API will remove a registered security endpoint identified by an endpoint name.

**参数**

• **pc**: [in] Pointer to the protocomm instance
• **ep_name**: [in] Endpoint identifier(name) string

**返回**

• ESP_OK: Success
• ESP_FAIL: Error adding endpoint / Endpoint with this name already exists
• ESP_ERR_INVALID_STATE: Security endpoint already set
• ESP_ERR_NO_MEM: Error allocating endpoint resource
• ESP_ERR_INVALID_ARG: Null instance/name/handler arguments

```c
esp_err_t protocomm_unset_security(protocomm_t *pc, const char* ep_name)
```
**esp_err_t** `protocomm_set_version(protocomm_t *pc, const char *ep_name, const char *version)`

Set endpoint for version verification.

This API can be used for setting an application specific protocol version which can be verified by clients through the endpoint.

**备注:**

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

---

### 参数

- `pc` - [in] Pointer to the protocomm instance
- `ep_name` - [in] Endpoint identifier (name) string
- `version` - [in] Version identifier (name) string

### 返回

- ESP_OK: Success
- ESP_FAIL: Error adding endpoint | Endpoint with this name already exists
- ESP_ERR_INVALID_STATE: Version endpoint already set
- ESP_ERR_NO_MEM: Error allocating endpoint resource
- ESP_ERR_INVALID_ARG: Null instance/name/handler arguments

**esp_err_t** `protocomm_unset_version(protocomm_t *pc, const char *ep_name)`

Remove version verification endpoint from a protocomm instance.

This API will remove a registered version endpoint identified by an endpoint name.

### 参数

- `pc` - [in] Pointer to the protocomm instance
- `ep_name` - [in] Endpoint identifier (name) string

### 返回

- ESP_OK: Success
- ESP_ERR_NOT_FOUND: Endpoint with specified name doesn’t exist
- ESP_ERR_INVALID_ARG: Null instance/name arguments

---

### Type Definitions

typedef **esp_err_t** (*`protocomm_req_handler_t`)(uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)

Function prototype for protocomm endpoint handler.

typedef struct protocomm **protocomm_t**

This structure corresponds to a unique instance of protocomm returned when the API `protocomm_new()` is called. The remaining Protocomm APIs require this object as the first parameter.

**备注:** Structure of the protocomm object is kept private

---

### Header File

- components/protocomm/include/security/protocomm_security.h

### Structures

- **protocomm_security1_params**

  Protocomm Security 1 parameters: Proof Of Possession.
Public Members

const uint8_t *data
    Pointer to buffer containing the proof of possession data

uint16_t len
    Length (in bytes) of the proof of possession data

struct protocomm_security2_params
    Protocomm Security 2 parameters: Salt and Verifier.

Public Members

const char *salt
    Pointer to the buffer containing the salt

uint16_t salt_len
    Length (in bytes) of the salt

const char *verifier
    Pointer to the buffer containing the verifier

uint16_t verifier_len
    Length (in bytes) of the verifier

struct protocomm_security
    Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

备注: This structure should not have any dynamic members to allow re-entrancy

Public Members

int ver
    Unique version number of security implementation

esp_err_t (*init)(protocomm_security_handle_t *handle)
    Function for initializing/allocating security infrastructure

esp_err_t (*cleanup)(protocomm_security_handle_t handle)
    Function for deallocating security infrastructure

esp_err_t (*new_transport_session)(protocomm_security_handle_t handle, uint32_t session_id)
    Starts a new secure transport session with specified ID

esp_err_t (*close_transport_session)(protocomm_security_handle_t handle, uint32_t session_id)
    Closes a secure transport session with specified ID
esp_err_t (*security_req_handler)(protocomm_security_handle_t handle, const void *sec_params, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)

Handler function for authenticating connection request and establishing secure session

esp_err_t (*encrypt)(protocomm_security_handle_t handle, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen)

Function which implements the encryption algorithm

esp_err_t (*decrypt)(protocomm_security_handle_t handle, uint32_t session_id, const uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen)

Function which implements the decryption algorithm

Type Definitions

typedef struct protocomm_security1_params protocomm_security1_params_t

Protocomm Security 1 parameters: Proof Of Possession.

typedef protocomm_security1_params_t protocomm_security_pop_t

typedef struct protocomm_security2_params protocomm_security2_params_t

Protocomm Security 2 parameters: Salt and Verifier.

typedef void *protocomm_security_handle_t

protocomm_security_t

Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

备注: This structure should not have any dynamic members to allow re-entrancy

Header File

- components/protocomm/include/security/protocomm_security0.h

Header File

- components/protocomm/include/security/protocomm_security1.h

Header File

- components/protocomm/include/transports/protocomm_httpd.h

Functions

esp_err_t protocomm_httpd_start (protocomm_t *pc, const protocomm_httpd_config_t *config)

Start HTTPD protocomm transport.

This API internally creates a framework to allow endpoint registration and security configuration for the protocomm.
This is a singleton. ie. Protocomm can have multiple instances, but only one instance can be bound to an HTTP transport layer.

**Parameters**
- `pc` [in] Protocomm instance pointer obtained from `protocomm_new()`
- `config` [in] Pointer to config structure for initializing HTTP server

**Return**
- `ESP_OK`: Success
- `ESP_ERR_INVALID_ARG`: Null arguments
- `ESP_ERR_NOT_SUPPORTED`: Transport layer bound to another protocomm instance
- `ESP_ERR_INVALID_STATE`: Transport layer already bound to this protocomm instance
- `ESP_ERR_NO_MEM`: Memory allocation for server resource failed
- `ESP_ERR_HTTPD_*`: HTTP server error on start

```c
esp_err_t protocomm_httpd_stop(protocomm_t *pc)
```

Stop HTTPD protocomm transport.

This API cleans up the HTTPD transport protocomm and frees all the handlers registered with the protocomm.

**Parameters**
- `pc` [in] Same protocomm instance that was passed to `protocomm_httpd_start()`

**Return**
- `ESP_OK`: Success
- `ESP_ERR_INVALID_ARG`: Null / incorrect protocomm instance pointer

**Unions**

```c
union protocomm_httpd_config_data_t
```

`#include <protocomm_httpd.h>` Protocomm HTTPD Configuration Data

**Public Members**

```c
void *handle
```

HTTP Server Handle, if ext_handle_provided is set to true

```c
protocomm_http_server_config_t config
```

HTTP Server Configuration, if a server is not already active

**Structures**

```c
struct protocomm_http_server_config_t
```

Config parameters for protocomm HTTP server.

**Public Members**

```c
uint16_t port
```

Port on which the HTTP server will listen

```c
size_t stack_size
```

Stack size of server task, adjusted depending upon stack usage of endpoint handler
unsigned task_priority
Priority of server task

struct protocomm_httpd_config_t
Config parameters for protocomm HTTP server.

Public Members

bool ext_handle_provided
Flag to indicate if an external HTTP Server Handle has been provided. In such a case, protocomm will use the same HTTP Server and not start a new one internally.

protocomm_httpd_config_data_t data
Protocomm HTTPD Configuration Data

Macros

PROTOCOMM_HTTPD_DEFAULT_CONFIG()

Header File

• components/protocomm/include/transports/protocomm_ble.h

Functions

esp_err_t protocomm_ble_start (protocomm_t *pc, const protocomm_ble_config_t *config)
Start Bluetooth Low Energy based transport layer for provisioning.
Initialize and start required BLE service for provisioning. This includes the initialization for characteristics/service for BLE.

参数
• pc [in] Protocomm instance pointer obtained from protocomm_new()
• config [in] Pointer to config structure for initializing BLE

返回
• ESP_OK: Success
• ESP_FAIL: Simple BLE start error
• ESP_ERR_NO_MEM: Error allocating memory for internal resources
• ESP_ERR_INVALID_STATE: Error in ble config
• ESP_ERR_INVALID_ARG: Null arguments

esp_err_t protocomm_ble_stop (protocomm_t *pc)
Stop Bluetooth Low Energy based transport layer for provisioning.
Stops service/task responsible for BLE based interactions for provisioning

备注: You might want to optionally reclaim memory from Bluetooth. Refer to the documentation of esp_bt_mem_release in that case.

参数 pc [in] Same protocomm instance that was passed to protocomm_ble_start()

返回
• ESP_OK: Success
• ESP_FAIL: Simple BLE stop error
• ESP_ERR_INVALID_ARG: Null / incorrect protocomm instance
Structures

struct name_uuid

This structure maps handler required by protocomm layer to UUIDs which are used to uniquely identify BLE characteristics from a smartphone or a similar client device.

Public Members

cnst char *name
    Name of the handler, which is passed to protocomm layer

tuint16_t uuid
    UUID to be assigned to the BLE characteristic which is mapped to the handler

struct protocomm_ble_config

Config parameters for protocomm BLE service.

Public Members

char device_name[MAX_BLE_DEVNAME_LEN + 1]
    BLE device name being broadcast at the time of provisioning

tuint8_t service_uuid[BLE_UUID128_VAL_LENGTH]
    128 bit UUID of the provisioning service

tuint8_t *manufacturer_data
    BLE device manufacturer data pointer in advertisement

 ssize_t manufacturer_data_len
    BLE device manufacturer data length in advertisement

 ssize_t nu_lookup_count
    Number of entries in the Name-UUID lookup table

struct protocomm_ble_name_uuid_t *nu_lookup
    Pointer to the Name-UUID lookup table

unsigned ble_bonding
    BLE bonding

unsigned ble_sm_sc
    BLE security flag

Macros

MAX_BLE_DEVNAME_LEN
    BLE device name cannot be larger than this value 31 bytes (max scan response size) - 1 byte (length) - 1 byte (type) = 29 bytes
**Chapter 2. API 参考**

**BLE_UUID128_VAL_LENGTH**

**MAX_BLE_MANUFACTURER_DATA_LEN**

Theoretically, the limit for max manufacturer length remains same as BLE device name i.e. 31 bytes (max scan response size) - 1 byte (length) - 1 byte (type) = 29 bytes However, manufacturer data goes along with BLE device name in scan response. So, it is important to understand the actual length should be smaller than (29 - (BLE device name length) - 2).

**Type Definitions**

typedef struct name_uuid protocomm_ble_name_uuid_t

This structure maps handler required by protocomm layer to UUIDs which are used to uniquely identify BLE characteristics from a smartphone or a similar client device.

typedef struct protocomm_ble_config protocomm_ble_config_t

Config parameters for protocomm BLE service.

**Enumerations**

enum protocomm_transport_ble_event_t

Events generated by BLE transport.

These events are generated when the BLE transport is paired and disconnected.

*Values:

enumerator PROTOCOL_TRANSPORT_BLE_CONNECTED

enumerator PROTOCOL_TRANSPORT_BLE_DISCONNECTED

### 2.8.2 Unified Provisioning

**Overview**

Unified provisioning support in the ESP-IDF provides an extensible mechanism to the developers to configure the device with the Wi-Fi credentials and/or other custom configuration using various transports and different security schemes. Depending on the use-case it provides a complete and ready solution for Wi-Fi network provisioning along with example iOS and Android applications. Or developers can extend the device-side and phone-app side implementations to accommodate their requirements for sending additional configuration data. Following are the important features of this implementation.

1. **Extensible Protocol**: The protocol is completely flexible and it offers the ability for the developers to send custom configuration in the provisioning process. The data representation too is left to the application to decide.
2. **Transport Flexibility**: The protocol can work on Wi-Fi (SoftAP + HTTP server) or on BLE as a transport protocol. The framework provides an ability to add support for any other transport easily as long as command-response behaviour can be supported on the transport.
3. **Security Scheme Flexibility**: It’s understood that each use-case may require different security scheme to secure the data that is exchanged in the provisioning process. Some applications may work with SoftAP that’s WPA2 protected or BLE with “just-works” security. Or the applications may consider the transport to be insecure and may want application level security. The unified provisioning framework allows application to choose the security as deemed suitable.
4. **Compact Data Representation**: The protocol uses Google Protobufs as a data representation for session setup and Wi-Fi provisioning. They provide a compact data representation and ability to parse the data in multiple programming languages in native format. Please note that this data representation is not forced on application specific data and the developers may choose the representation of their choice.
Typical Provisioning Process

Deciding on Transport

Unified provisioning subsystem supports Wi-Fi (SoftAP+HTTP server) and BLE (GATT based) transport schemes. Following points need to be considered while selecting the best possible transport for provisioning.

1. BLE based transport has an advantage that in the provisioning process, the BLE communication channel stays intact between the device and the client. That provides reliable provisioning feedback.
2. BLE based provisioning implementation makes the user-experience better from the phone apps as on Android and iOS both, the phone app can discover and connect to the device without requiring user to go out of the phone app.
3. BLE transport however consumes ~110KB memory at runtime. If the product does not use the BLE or BT functionality after provisioning is done, almost all the memory can be reclaimed back and can be added into the heap.
4. SoftAP based transport is highly interoperable; however as the same radio is shared between SoftAP and Station interface, the transport is not reliable in the phase when the Wi-Fi connection to external AP is attempted. Also, the client may roam back to different network when the SoftAP changes the channel at the time of Station connection.
5. SoftAP transport does not require much additional memory for the Wi-Fi use-cases
6. SoftAP based provisioning requires the phone app user to go to “System Settings” to connect to Wi-Fi network hosted by the device in case of iOS. The discovery (scanning) as well as connection API is not available for the iOS applications.

Deciding on Security

Depending on the transport and other constraints the security scheme needs to be selected by the application developers. Following considerations need to be given from the provisioning security perspective: 1. The configuration data sent from the client to the device and the response has to be secured. 2. The client should authenticate the device it is connected to. 3. The device manufacturer may choose proof-of-possession - a unique per device secret to be entered on the provisioning client as a security measure to make sure that the user can provisions the device in the possession.

There are two levels of security schemes. The developer may select one or combination depending on requirements.

1. Transport Security: SoftAP provisioning may choose WPA2 protected security with unique per-device passphrase. Per-device unique passphrase can also act as a proof-of-possession. For BLE, “just-works” security can be used as a transport level security after understanding the level of security it provides.
2. Application Security: The unified provisioning subsystem provides application level security (security1) that provides data protection and authentication (through proof-of-possession) if the application does not use the transport level security or if the transport level security is not sufficient for the use-case.

Device Discovery

The advertisement and device discovery is left to the application and depending on the protocol chosen, the phone apps and device firmware application can choose appropriate method to advertise and discovery.

For the SoftAP+HTTP transport, typically the SSID (network name) of the AP hosted by the device can be used for discovery.

For the BLE transport device name or primary service included in the advertisement or combination of both can be used for discovery.

Architecture

The below diagram shows architecture of unified provisioning.
Chapter 2. API

1. Transport specific discovery and connection
   - Some form of beaconing
   - Client connects

2. Session Establishment
   - Get Version Request
     - Get Version Response
   - Session Setup Request
     - Session Setup Response

One or multiple steps as per protocol

3. Configuration
   - App specific Set Config (optional)
     - Set Config Response (optional)
   - Wi-Fi SetConfig(SSID, Passphrase...)
     - Wi-Fi SetConfig response
   - Wi-Fi ApplyConfig cmd
     - Wi-Fi ApplyConfig resp
   - Wi-Fi GetStatus cmd (repeated)
     - Wi-Fi GetStatus resp (repeated)

4. Close connection
   - Close Connection
Chapter 2. API

It relies on the base layer called **Protocol Communication** (Protocol Communication) which provides a framework for security schemes and transport mechanisms. Wi-Fi Provisioning layer uses Protocomm to provide simple callbacks to the application for setting the configuration and getting the Wi-Fi status. The application has control over implementation of these callbacks. In addition application can directly use protocomm to register custom handlers.

Application creates a protocomm instance which is mapped to a specific transport and specific security scheme. Each transport in the protocomm has a concept of an “end-point” which corresponds to logical channel for communication for specific type of information. For example security handshake happens on a different endpoint than the Wi-Fi configuration endpoint. Each end-point is identified using a string and depending on the transport internal representation of the end-point changes. In case of SoftAP+HTTP transport the end-point corresponds to URI whereas in case of BLE the end-point corresponds to GATT characteristic with specific UUID. Developers can create custom end-points and implement handler for the data that is received or sent over the same end-point.

**Security Schemes**

At present, unified provisioning supports the following security schemes:

1. **Security0** - No security (No encryption)
2. **Security1** - Curve25519-based key exchange, shared key derivation and AES256-CTR mode encryption of the data. It supports two modes:
   a. Authorized - Proof of Possession (PoP) string used to authorize session and derive shared key
   b. No Auth (Null PoP) - Shared key derived through key exchange only
3. **Security2** - SRP6a-based shared key derivation and AES256-GCM mode encryption of the data.

The respective security schemes need to be enabled through the project configuration menu. Please refer to the Enabling protocomm security version section in **Protocol Communication** (Protocol Communication) for more details.
Security Scheme

Security scheme details are shown in the below sequence diagram -

Security2 Scheme

Security2 scheme is based on the Secure Remote Password (SRP6a) protocol - RFC 5054. The protocol requires the Salt and Verifier to be generated beforehand with help of the identifying username `I` and the plaintext password `p`. The Salt and Verifier are then stored on ESP32. The password `p` and username `I` are to be provided to the Phone App (Provisioning entity) by suitable means for example QR code sticker.

Security2 scheme details are shown in the below sequence diagram -

Sample Code

Please refer to Protocol Communication and Wi-Fi Provisioning for API guides and code snippets on example usage.

Application implementation can be found as an example under provisioning.

Provisioning Tools

Provisioning applications are available for various platforms, along with source code:

• Android:
  – BLE Provisioning app on Play Store.
  – SoftAP Provisioning app on Play Store.
  – Source code on GitHub: esp-idf-provisioning-android.
• iOS:
  – BLE Provisioning app on app store.
  – SoftAP Provisioning app on app Store.
  – Source code on GitHub: esp-idf-provisioning-ios.
• Linux/MacOS/Windows: tools/esp_prov (a python based command line tool for provisioning)

The phone applications offer simple UI and thus more user centric, while the command line application is useful as a debugging tool for developers.

2.8.3 Wi-Fi Provisioning

Overview

This component provides APIs that control Wi-Fi provisioning service for receiving and configuring Wi-Fi credentials over SoftAP or BLE transport via secure Protocol Communication (protocomm) sessions. The set of wifi_prov_mgr APIs help in quickly implementing a provisioning service having necessary features with minimal amount of code and sufficient flexibility.

Initialization  

`wifi_prov_mgr_init()` is called to configure and initialize the provisioning manager and thus this must be called prior to invoking any other wifi_prov_mgr APIs. Note that the manager relies on other components of IDF, namely NVS, TCP/IP, Event Loop and Wi-Fi (and optionally mDNS), hence these must be initialized beforehand. The manager can be de-initialized at any moment by making a call to `wifi_prov_mgr_deinit()`.

```c
wifi_prov_mgr_config_t config = {
  .scheme = wifi_prov_scheme_ble,
  .scheme_event_handler = WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
};
ESP_ERROR_CHECK( wifi_prov_mgr_init(config) );
```
Chapter 2. API

Security

---

**Client**

Generate Key Pair

\[(\text{cli\_privkey}, \text{cli\_pubkey}) = \text{curve25519\_keygen}()\]

SessionCmd0(\text{cli\_pubkey})

---

**Device**

Generate Key Pair

\[(\text{dev\_privkey}, \text{dev\_pubkey}) = \text{curve25519\_keygen}()\]

Initialization Vector

\[\text{dev\_rand} = \text{gen\_16\_byte\_random}()\]

Shared Key

\[\text{shared\_key(No PoP)} = \text{curve25519}(\text{dev\_privkey}, \text{cli\_pubkey})\]

\[\text{shared\_key(with PoP)} = \text{curve25519}(\text{dev\_privkey}, \text{cli\_pubkey}) \oplus \text{SHA256}(\text{pop})\]

SessionResp0(\text{dev\_pubkey, dev\_rand})

---

Shared Key

\[\text{shared\_key(No PoP)} = \text{curve25519}(\text{cli\_privkey, dev\_pubkey})\]

\[\text{shared\_key(with PoP)} = \text{curve25519}(\text{cli\_privkey, dev\_pubkey}) \oplus \text{SHA256}(\text{pop})\]

---

Verification Token

\[\text{cli\_verify} = \text{aes\_ctr\_enc}(\text{key}=\text{shared\_key, data} = \text{dev\_pubkey, nonce}=\text{dev\_rand})\]

SessionCmd1(\text{cli\_verify})

---

Verify Client

\[\text{check (dev\_pubkey} = \text{aes\_ctr\_dec(cli\_verify...)}\]

\[\text{dev\_verify} = \text{aes\_ctr\_enc}(\text{key}=\text{shared\_key, data} = \text{cli\_pubkey, nonce}=(\text{prev-context}))\]

SessionResp1(\text{dev\_verify})

---

Verify Device

\[\text{check (cli\_pubkey} = \text{aes\_ctr\_dec(dev\_verify...)}\]

图 28: Security1
Chapter 2. API

Generate Key Pair

Client (PhoneApp)

Device (ESP)

a (cli_privkey) = 256 bit random value, A (cli_pubkey) = g^a.
g - generator, N - large safe prime.
All arithmetic operations are performed in ring of integers modulo N,
thus all occurrences like y^z should be read as y^z modulo N.

SessionCmd0(cli_pubkey A, username)

Obtain salt and verifier stored on esp
Salt s = 256 bit random value,
Verifier v = g^x where x = H(s || p)

b (dev_privkey) = 256 bit random value
B (dev_pubkey) = k*v + g^b where k = H(N, g)

Shared Key K = H(S) where,
S = (A*v^u) + b
u = H(A, B)

SessionResp0(dev_pubkey B, dev_rand)

shared_key(K) = H(S) where,
S = (B - k^y) + (a + u)
K = H(A, B),
v = g^x,
x = H(s || p).

Verification Token

client_proof M = H(H(N) XOR H(g) || H(l) || s || A || B || K)

SessionCmd1(client_proof M)

device generates M1 = H(H(N) XOR H(g) || H(l) || s || A || B || K)
device verifies this M1 with the M1 obtained from Client

Device generate device_proof M2 = H(A, M, K)

dev_rand = gen_16byte_random() This random number is to be used for AES-GCM operation
for encryption and decryption of data using the shared secret

SessionResp1(device_proof M2, dev_rand)

Verify Device

Client calculates device proof M2 as M2 = H(A, M, K)
client verifies this M2 with M2 obtained from device

Submit Document Feedback
The configuration structure `wifi_prov_mgr_config_t` has a few fields to specify the behavior desired of the manager:

- **scheme**: This is used to specify the provisioning scheme. Each scheme corresponds to one of the modes of transport supported by protocomm. Hence, we have three options:
  - `wifi_prov_scheme_ble`: BLE transport and GATT Server for handling provisioning commands
  - `wifi_prov_scheme_softap`: Wi-Fi SoftAP transport and HTTP Server for handling provisioning commands
  - `wifi_prov_scheme_console`: Serial transport and console for handling provisioning commands

- **scheme_event_handler**: An event handler defined along with scheme. Choosing appropriate scheme specific event handler allows the manager to take care of certain matters automatically. Presently this is not used for either SoftAP or Console based provisioning, but is very convenient for BLE. To understand how, we must recall that Bluetooth requires quite some amount of memory to function and once provisioning is finished, the main application may want to reclaim back this memory (or part of it, if it needs to use either BLE or classic BT). Also, upon every future re-boot of a provisioned device, this reclamation of memory needs to be performed again. To reduce this complication in using `wifi_prov_scheme_ble`, the scheme specific handlers have been defined, and depending upon the chosen handler, the BLE / classic BT / BTDM memory will be freed automatically when the provisioning manager is de-initialized. The available options are:
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM`: Free both classic BT and BLE (BTDM) memory. Used when main application doesn’t require Bluetooth at all.
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE`: Free only BLE memory. Used when main application requires classic BT.
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT`: Free only classic BT. Used when main application requires BLE. In this case freeing happens right when the manager is initialized.
  - `WIFI_PROV_EVENT_HANDLER_NONE`: Don’t use any scheme specific handler. Used when provisioning scheme is not BLE (i.e. SoftAP or Console), or when main application wants to handle the memory reclaiming on its own, or needs both BLE and classic BT to function.

- **app_event_handler** (Deprecated): It is now recommended to catch `WIFI_PROV_EVENT`’s that are emitted to the default event loop handler. See definition of `wifi_prov_cb_event_t` for the list of events that are generated by the provisioning service. Here is an excerpt showing some of the provisioning events:

```c
static void event_handler(void* arg, esp_event_base_t event_base, int event_id, void* event_data) {
    if (event_base == WIFI_PROV_EVENT) {
        switch (event_id) {
            case WIFI_PROV_START:
                ESP_LOGI(TAG, "Provisioning started");
                break;
            case WIFI_PROV_CRED_RECV:
                wifi_sta_config_t *wifi_sta_cfg = (wifi_sta_config_t*)event_data;
                ESP_LOGI(TAG, "Received Wi-Fi credentials\nSSID : %s\nPassword : %s",
                        wifi_sta_cfg->ssid,
                        wifi_sta_cfg->password);
                break;
            case WIFI_PROV_CRED_FAIL:
                wifi_prov_sta_fail_reason_t *reason = (wifi_prov_sta_fail_reason_t*)event_data;
                ESP_LOGE(TAG, "Provisioning failed!\nReason : %s\nPlease reset to factory and retry\nprovisioning",
```

(下页继续)
Chapter 2. API 参考

(*reason == WIFI_prov_STA_AUTH_ERROR) ?
"Wi-Fi station authentication failed" : 
"access-point not found")
break;
}
case WIFI_prov_CRED_SUCCESS:
    ESP_LOGI(TAG, "Provisioning successful");
break;
case WIFI_prov_END:
    /* De-initialize manager once provisioning is finished */
    wifi_prov_mgr_deinit();
break;
default:
    break;
}
}

The manager can be de-initialized at any moment by making a call to wifi_prov_mgr_deinit().

Check Provisioning State  Whether device is provisioned or not can be checked at runtime by calling wifi_prov_mgr_is_provisioned(). This internally checks if the Wi-Fi credentials are stored in NVS.

Note that presently manager does not have its own NVS namespace for storage of Wi-Fi credentials, instead it relies on the esp_wifi_APIs to set and get the credentials stored in NVS from the default location.

If provisioning state needs to be reset, any of the following approaches may be taken:

- the associated part of NVS partition has to be erased manually
- main application must implement some logic to call esp_wifi_APIs for erasing the credentials at runtime
- main application must implement some logic to force start the provisioning irrespective of the provisioning state

```c
bool provisioned = false;
ESP_ERROR_CHECK( wifi_prov_mgr_is_provisioned(&provisioned) );
```

Start Provisioning Service  At the time of starting provisioning we need to specify a service name and the corresponding key. These translate to:

- Wi-Fi SoftAP SSID and passphrase, respectively, when scheme is wifi_prov_scheme_softap
- BLE Device name (service key is ignored) when scheme is wifi_prov_scheme_ble

Also, since internally the manager uses protocomm, we have the option of choosing one of the security features provided by it:

- Security 1 is secure communication which consists of a prior handshake involving X25519 key exchange along with authentication using a proof of possession (pop), followed by AES-CTR for encryption/decryption of subsequent messages
- Security 0 is simply plain text communication. In this case the pop is simply ignored

See Provisioning for details about the security features.

```c
const char *service_name = "my_device";
const char *service_key = "password";

wifi_prov_security_t security = WIFI_PROV_SECURITY_1;
const char *pop = "abcd1234";

ESP_ERROR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_--name, service_key) );
```
The provisioning service will automatically finish only if it receives valid Wi-Fi AP credentials followed by successfully connection of device to the AP (IP obtained). Regardless of that, the provisioning service can be stopped at any moment by making a call to `wifi_prov_mgr_stop_provisioning()`.

**备注:** If the device fails to connect with the provided credentials, it won’t accept new credentials anymore, but the provisioning service will keep on running (only to convey failure to the client), until the device is restarted. Upon restart the provisioning state will turn out to be true this time (as credentials will be found in NVS), but device will again fail to connect with those same credentials (unless an AP with the matching credentials somehow does become available). This situation can be fixed by resetting the credentials in NVS or force starting the provisioning service. This has been explained above in Check Provisioning State.

### Waiting For Completion

Typically, the main application will wait for the provisioning to finish, then de-initialize the manager to free up resources and finally start executing its own logic.

There are two ways for making this possible. The simpler way is to use a blocking call to `wifi_prov_mgr_wait()`.

```c
// Start provisioning service
ESP_ERROR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_name, service_key) );

// Wait for service to complete
wifi_prov_mgr_wait();

// Finally de-initialize the manager
wifi_prov_mgr_deinit();
```

The other way is to use the default event loop handler to catch WIFI_PROV_EVENT’s and call :cpp:func:`wifi_prov_mgr_deinit()` when event ID is `WIFI_PROV_END`:

```c
static void event_handler(void* arg, esp_event_base_t event_base, int event_id, void* event_data)
{
    if (event_base == WIFI_PROV_EVENT && event_id == WIFI_PROV_END) {
        /* De-initialize manager once provisioning is finished */
        wifi_prov_mgr_deinit();
    }
}
```

### User Side Implementation

When the service is started, the device to be provisioned is identified by the advertised service name which, depending upon the selected transport, is either the BLE device name or the SoftAP SSID. When using SoftAP transport, for allowing service discovery, mDNS must be initialized before starting provisioning. In this case the hostname set by the main application is used, and the service type is internally set to `_esp_wifi_prov`.

When using BLE transport, a custom 128 bit UUID should be set using `wifi_prov_scheme_ble_set_service_uuid()`. This UUID will be included in the BLE advertisement and will correspond to the primary GATT service that provides provisioning endpoints as GATT characteristics. Each GATT characteristic will be formed using the primary service UUID as base, with different auto assigned 12th and 13th bytes (assume counting starts from 0th byte). Since, an endpoint characteristic UUID is auto assigned, it shouldn’t be used to identify the endpoint. Instead, client side applications should identify the endpoints by reading the User Characteristic Description (0x2901) descriptor for each characteristic, which contains the endpoint name of the characteristic. For example, if the service UUID is set to `55cc035e-fb27-4f80-be02-3c60828b7451`, each endpoint characteristic will be assigned a UUID like `55cc____-fb27-4f80-be02-3c60828b7451`, with unique values at the 12th and 13th bytes.

Once connected to the device, the provisioning related protocomm endpoints can be identified as follows:
### Table 9: Endpoints provided by Provisioning Service

<table>
<thead>
<tr>
<th>Endpoint Name (BLE + GATT Server)</th>
<th>URI (SoftAP + HTTP Server + mDNS)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prov-session</td>
<td>http://&lt;mdns-hostname&gt;.local/prov-session</td>
<td>Security endpoint used for session establishment</td>
</tr>
<tr>
<td>prov-scan</td>
<td><a href="http://wifi-prov.local/prov-scan">http://wifi-prov.local/prov-scan</a></td>
<td>Endpoint used for starting Wi-Fi scan and receiving scan results</td>
</tr>
<tr>
<td>prov-ctrl</td>
<td><a href="http://wifi-prov.local/prov-ctrl">http://wifi-prov.local/prov-ctrl</a></td>
<td>Endpoint used for controlling Wi-Fi provisioning state</td>
</tr>
<tr>
<td>prov-config</td>
<td>http://&lt;mdns-hostname&gt;.local/prov-config</td>
<td>Endpoint used for configuring Wi-Fi credentials on device</td>
</tr>
<tr>
<td>proto-ver</td>
<td>http://&lt;mdns-hostname&gt;.local/proto-ver</td>
<td>Endpoint for retrieving version info</td>
</tr>
</tbody>
</table>

Immediately after connecting, the client application may fetch the version / capabilities information from the `proto-ver` endpoint. All communications to this endpoint are un-encrypted, hence necessary information (that may be relevant for deciding compatibility) can be retrieved before establishing a secure session. The response is in JSON format and looks like: `prov: { ver: v1.1, cap: [no_pop] }, my_app: { ver: 1.345, cap: [cloud, local_ctrl] }, ....` Here label `prov` provides provisioning service version (`ver`) and capabilities (`cap`). For now, only `no_pop` capability is supported, which indicates that the service doesn’t require proof of possession for authentication. Any application related version / capabilities will be given by other labels (like `my_app` in this example). These additional fields are set using `wifi_prov_mgr_set_app_info()`.

User side applications need to implement the signature handshaking required for establishing and authenticating secure protocomm sessions as per the security scheme configured for use (this is not needed when manager is configured to use protocomm security 0).

See Unified Provisioning for more details about the secure handshake and encryption used. Applications must use the `.proto` files found under `protocomm/proto`, which define the Protobuf message structures supported by `prov-session` endpoint.

Once a session is established, Wi-Fi credentials are configured using the following set of `wifi_config` commands, serialized as Protobuf messages (the corresponding `.proto` files can be found under `wifi_provisioning/proto`):

- **get_status** - For querying the Wi-Fi connection status. The device will respond with a status which will be one of connecting / connected / disconnected. If status is disconnected, a disconnection reason will also be included in the status response.
- **set_config** - For setting the Wi-Fi connection credentials
- **apply_config** - For applying the credentials saved during `set_config` and start the Wi-Fi station

After session establishment, client can also request Wi-Fi scan results from the device. The results returned is a list of AP SSIDs, sorted in descending order of signal strength. This allows client applications to display APs nearby to the device at the time of provisioning, and users can select one of the SSIDs and provide the password which is then sent using the `wifi_config` commands described above. The `wifi_scan` endpoint supports the following protobuf commands:

- **scan_start** - For starting Wi-Fi scan with various options:
  - **blocking** (input) - If true, the command returns only when the scanning is finished
  - **passive** (input) - If true scan is started in passive mode (this may be slower) instead of active mode
  - **group_channels** (input) - This specifies whether to scan all channels in one go (when zero) or perform scanning of channels in groups, with 120ms delay between scanning of consecutive groups, and the value of this parameter sets the number of channels in each group. This is useful when transport mode is SoftAP, where scanning all channels in one go may not give the Wi-Fi driver enough time to send out beacons, and hence may cause disconnection with any connected stations. When scanning in groups, the manager will wait for atleast 120ms after completing scan on a group of channels, and thus allow the driver to send out the beacons. For example, given that the total number of Wi-Fi channels is 14, then setting `group_channels` to 4, will create 5 groups, with each group having 3 channels, except the last one
which will have \( 14 \% 3 = 2 \) channels. So, when scan is started, the first 3 channels will be scanned, 
followed by a 120ms delay, and then the next 3 channels, and so on, until all the 14 channels have been 
escanned. One may need to adjust this parameter as having only few channels in a group may slow down 
the overall scan time, while having too many may again cause disconnection. Usually a value of 4 should 
work for most cases. Note that for any other mode of transport, e.g. BLE, this can be safely set to 0, and 
hence achieve the fastest overall scanning time.

- **period_ms** (input) - Scan parameter specifying how long to wait on each channel
- **scan_status** - Gives the status of scanning process:
  - **scan_finished** (output) - When scan has finished this returns true
  - **result_count** (output) - This gives the total number of results obtained till now. If scan is yet happening 
    this number will keep on updating
- **scan_result** - For fetching scan results. This can be called even if scan is still on going
  - **start_index** (input) - Starting index from where to fetch the entries from the results list
  - **count** (input) - Number of entries to fetch from the starting index
  - **entries** (output) - List of entries returned. Each entry consists of ssid, channel and rssi information

The client can also control the provisioning state of the device using `wifi_ctrl` endpoint. The `wifi_ctrl` endpoint supports 
the following protobuf commands:

- **ctrl_reset** - Resets internal state machine of the device and clears provisioned credentials only in case of pro-
  visioning failures.
- **ctrl_reprov** - Resets internal state machine of the device and clears provisioned credentials only in case the 
device is to be provisioned again for new credentials after a previous successful provisioning

### Additional Endpoints

In case users want to have some additional protocomm endpoints customized to their re-
quirements, this is done in two steps. First is creation of an endpoint with a specific name, and the second step is 
the registration of a handler for this endpoint. See `protocomm` for the function signature of an endpoint handler. 
A custom endpoint must be created after initialization and before starting the provisioning service. Whereas, the 
protocomm handler is registered for this endpoint only after starting the provisioning service.

```c
wifi_prov_mgr_init(config);
wifi_prov_mgr_endpoint_create("custom-endpoint");
wifi_prov_mgr_start_provisioning(security, pop, service_name, service_
    --key);
wifi_prov_mgr_endpoint_register("custom-endpoint", custom_ep_handler,
    --custom_ep_data);
```

When the provisioning service stops, the endpoint is unregistered automatically.

One can also choose to call `wifi_prov_mgr_endpoint_unregister()` to manually deactivate an endpoint 
at runtime. This can also be used to deactivate the internal endpoints used by the provisioning service.

### When / How To Stop Provisioning Service?

The default behavior is that once the device successfully connects 
using the Wi-Fi credentials set by the `apply_config` command, the provisioning service will be stopped (and BLE 
/ SoftAP turned off) automatically after responding to the next `get_status` command. If `get_status` command is not 
received by the device, the service will be stopped after a 30s timeout.

On the other hand, if device was not able to connect using the provided Wi-Fi credentials, due to incorrect SSID / 
passphrase, the service will keep running, and `get_status` will keep responding with disconnected status and reason for 
disconnection. Any further attempts to provide another set of Wi-Fi credentials, will be rejected. These credentials 
will be preserved, unless the provisioning service is force started, or NVS erased.

If this default behavior is not desired, it can be disabled by calling `wifi_prov_mgr_disable_auto_stop()`. 
Now the provisioning service will only be stopped after an explicit call to 
`wifi_prov_mgr_stop_provisioning()`, which returns immediately after scheduling a task for stopping 
the service. The service stops after a certain delay and WIFI_PROV_END event gets emitted. This delay is specified 
by the argument to `wifi_prov_mgr_disable_auto_stop()`.

The customized behavior is useful for applications which want the provisioning service to be stopped some 
time after the Wi-Fi connection is successfully established. For example, if the application requires the de-
vice to connect to some cloud service and obtain another set of credentials, and exchange this credentials
over a custom protocomm endpoint, then after successfully doing so stop the provisioning service by calling `wifi_prov_mgr_stop_provisioning()` inside the protocomm handler itself. The right amount of delay ensures that the transport resources are freed only after the response from the protocomm handler reaches the client side application.

Application Examples

For complete example implementation see `provisioning/wifi_prov_mgr`

Provisioning Tools

Provisioning applications are available for various platforms, along with source code:

- **Android:**
  - BLE Provisioning app on Play Store.
  - SoftAP Provisioning app on Play Store.
  - Source code on GitHub: `esp-idf-provisioning-android`.

- **iOS:**
  - BLE Provisioning app on app store.
  - SoftAP Provisioning app on app store.
  - Source code on GitHub: `esp-idf-provisioning-ios`.

- **Linux/MacOS/Windows:** `tools/esp_prov` (a python based command line tool for provisioning)

The phone applications offer simple UI and thus more user centric, while the command line application is useful as a debugging tool for developers.

API Reference

Header File

- `components/wifi_provisioning/include/wifi_provisioning/manager.h`

Functions

```c
esp_err_t wifi_prov_mgr_init (wifi_prov_mgr_config_t config)
```

Initialize provisioning manager instance.

`config` is configuration structure

```c
void wifi_prov_mgr_deinit (void)
```

Stop provisioning (if running) and release resource used by the manager.

```c
esp_err_t wifi_prov_mgr_is_provisioned (bool *provisioned)
```

Checks if device is provisioned.

This checks if Wi-Fi credentials are present on the NVS

The Wi-Fi credentials are assumed to be kept in the same NVS namespace as used by esp_wifi component
If one were to call esp_wifi_set_config() directly instead of going through the provisioning process, this function will still yield true (i.e. device will be found to be provisioned).

备注： Calling wifi_prov_mgr_start_provisioning() automatically resets the provision state, irrespective of what the state was prior to making the call.

### 参数

- **provisioned** - [out] True if provisioned, else false
- **ESP_OK** : Retrieved provision state successfully
- **ESP_FAIL** : Wi-Fi not initialized
- **ESP_ERR_INVALID_ARG** : Null argument supplied

```c
esp_err_t wifi_prov_mgr_start_provisioning(wifi_prov_security_t security, const void *wifi_prov_sec_params, const char *service_name, const char *service_key)
```

Start provisioning service.

This starts the provisioning service according to the scheme configured at the time of initialization. For scheme:

- wifi_prov_scheme_ble : This starts protocomm_ble, which internally initializes BLE transport and starts GATT server for handling provisioning requests.
- wifi_prov_scheme_softap : This activates SoftAP mode of Wi-Fi and starts protocomm_httpd, which internally starts an HTTP server for handling provisioning requests (If mDNS is active it also starts advertising service with type _esp_wifi_prov._tcp)

Event WIFI_PROV_START is emitted right after provisioning starts without failure.

备注： This API will start provisioning service even if device is found to be already provisioned, i.e. wifi_prov_mgr_is_provisioned() yields true.

### 参数

- **security** - [in] Specify which protocomm security scheme to use:
  - WIFI_PROV_SECURITY_0 : For no security
  - WIFI_PROV_SECURITY_1 : x25519 secure handshake for session establishment followed by AES-CTR encryption of provisioning messages
  - WIFI_PROV_SECURITY_2 : SRP6a based authentication and key exchange followed by AES-GCM encryption/decryption of provisioning messages
- **wifi_prov_sec_params** - [in] Pointer to security params (NULL if not needed). This is not needed for protocomm security 0. This pointer should hold the struct of type wifi_prov_security1_params_t for protocomm security 1 and wifi_prov_security2_params_t for protocomm security 2 respectively. This pointer and its contents should be valid till the provisioning service is running and has not been stopped or de-inited.
- **service_name** - [in] Unique name of the service. This translates to:
  - Wi-Fi SSID when provisioning mode is softAP
  - Device name when provisioning mode is BLE
- **service_key** - [in] Key required by client to access the service (NULL if not needed). This translates to:
  - Wi-Fi password when provisioning mode is softAP
  - ignored when provisioning mode is BLE

### 返回

- **ESP_OK** : Provisioning started successfully
- **ESP_FAIL** : Failed to start provisioning service
- **ESP_ERR_INVALID_STATE** : Provisioning manager not initialized or already started
### Chapter 2. API 参考

**void wifi_prov_mgr_stop_provisioning (void)**

Stop provisioning service.

If provisioning service is active, this API will initiate a process to stop the service and return. Once the service actually stops, the event WIFI_PROV_END will be emitted.

If wifi_prov_mgr_deinit() is called without calling this API first, it will automatically stop the provisioning service and emit the WIFI_PROV_END, followed by WIFI_PROV_DEINIT, before returning.

This API will generally be used along with wifi_prov_mgr_disable_auto_stop() in the scenario when the main application has registered its own endpoints, and wishes that the provisioning service is stopped only when some protocomm command from the client side application is received.

Calling this API inside an endpoint handler, with sufficient cleanup_delay, will allow the response / acknowledgment to be sent successfully before the underlying protocomm service is stopped.

Cleanup_delay is set when calling wifi_prov_mgr_disable_auto_stop(). If not specified, it defaults to 1000ms.

For straightforward cases, using this API is usually not necessary as provisioning is stopped automatically once WIFI_PROV_CRED_SUCCESS is emitted. Stopping is delayed (maximum 30 seconds) thus allowing the client side application to query for Wi-Fi state, i.e. after receiving the first query and sending Wi-Fi state connected response the service is stopped immediately.

**void wifi_prov_mgr_wait (void)**

Wait for provisioning service to finish.

Calling this API will block until provisioning service is stopped i.e. till event WIFI_PROV_END is emitted.

This will not block if provisioning is not started or not initialized.

**esp_err_t wifi_prov_mgr_disable_auto_stop (uint32_t cleanup_delay)**

Disable auto stopping of provisioning service upon completion.

By default, once provisioning is complete, the provisioning service is automatically stopped, and all endpoints (along with those registered by main application) are deactivated.

This API is useful in the case when main application wishes to close provisioning service only after it receives some protocomm command from the client side app. For example, after connecting to Wi-Fi, the device may want to connect to the cloud, and only once that is successfully, the device is said to be fully configured. But, then it is upto the main application to explicitly call wifi_prov_mgr_stop_provisioning() later when the device is fully configured and the provisioning service is no longer required.

**备注:** This must be called before executing wifi_prov_mgr_start_provisioning()

参数 cleanup_delay —[in] Sets the delay after which the actual cleanup of transport related resources is done after a call to wifi_prov_mgr_stop_provisioning() returns. Minimum allowed value is 100ms. If not specified, this will default to 1000ms.

返回

- ESP_OK: Success
- ESP_ERR_INVALID_STATE: Manager not initialized or provisioning service already started

**esp_err_t wifi_prov_mgr_set_app_info (const char *label, const char *version, const char *capabilities, size_t total_capabilities)**

Set application version and capabilities in the JSON data returned by proto-ver endpoint.

This function can be called multiple times, to specify information about the various application specific services running on the device, identified by unique labels.

The provisioning service itself registers an entry in the JSON data, by the label “prov”, containing only provisioning service version and capabilities. Application services should use a label other than “prov” so as not to overwrite this.
备注：This must be called before executing wifi_prov_mgr_start_provisioning()

参数
- **label** [in] String indicating the application name.
- **version** [in] String indicating the application version. There is no constraint on format.
- **capabilities** [in] Array of strings with capabilities. These could be used by the client side app to know the application registered endpoint capabilities
- **total_capabilities** [in] Size of capabilities array

返回
- **ESP_OK**: Success
- **ESP_ERR_INVALID_STATE**: Manager not initialized or provisioning service already started
- **ESP_ERR_NO_MEM**: Failed to allocate memory for version string
- **ESP_ERR_INVALID_ARG**: Null argument

**esp_err_t wifi_prov_mgr_endpoint_create**(const char* ep_name)

Create an additional endpoint and allocate internal resources for it.

This API is to be called by the application if it wants to create an additional endpoint. All additional endpoints will be assigned UUIDs starting from 0xFF54 and so on in the order of execution.

protocomm handler for the created endpoint is to be registered later using wifi_prov_mgr_endpoint_register() after provisioning has started.

备注：This API can only be called BEFORE provisioning is started

备注：Additional endpoints can be used for configuring client provided parameters other than Wi-Fi credentials, that are necessary for the main application and hence must be set prior to starting the application

备注：After session establishment, the additional endpoints must be targeted first by the client side application before sending Wi-Fi configuration, because once Wi-Fi configuration finishes the provisioning service is stopped and hence all endpoints are unregistered

参数 **ep_name** [in] unique name of the endpoint

返回
- **ESP_OK**: Success
- **ESP_FAIL**: Failure

**esp_err_t wifi_prov_mgr_endpoint_register**(const char* ep_name, protocomm_req_handler_t handler, void* user_ctx)

Register a handler for the previously created endpoint.

This API can be called by the application to register a protocomm handler to any endpoint that was created using wifi_prov_mgr_endpoint_create().

备注：This API can only be called AFTER provisioning has started

备注：Additional endpoints can be used for configuring client provided parameters other than Wi-Fi credentials, that are necessary for the main application and hence must be set prior to starting the application
Chapter 2. API 参考

备注：After session establishment, the additional endpoints must be targeted first by the client side application before sending Wi-Fi configuration, because once Wi-Fi configuration finishes the provisioning service is stopped and hence all endpoints are unregistered.

### 参数
- **ep_name** – [in] Name of the endpoint
- **handler** – [in] Endpoint handler function
- **user_ctx** – [in] User data

### 返回
- **ESP_OK** : Success
- **ESP_FAIL** : Failure

```c
void wifi_prov_mgr_endpoint_unregister (const char* ep_name)
```

Unregister the handler for an endpoint.

This API can be called if the application wants to selectively unregister the handler of an endpoint while the provisioning is still in progress.

All the endpoint handlers are unregistered automatically when the provisioning stops.

### 参数 ep_name – [in] Name of the endpoint

```c
esp_err_t wifi_prov_mgr_get_wifi_state (wifi_prov_sta_state_t *state)
```

Get state of Wi-Fi Station during provisioning.

- **state** – [out] Pointer to wifi_prov_sta_state_t variable to be filled

### 返回
- **ESP_OK** : Successfully retrieved Wi-Fi state
- **ESP_FAIL** : Provisioning app not running

```c
esp_err_t wifi_prov_mgr_get_wifi_disconnect_reason (wifi_prov_sta_fail_reason_t *reason)
```

Get reason code in case of Wi-Fi station disconnection during provisioning.

- **reason** – [out] Pointer to wifi_prov_sta_fail_reason_t variable to be filled

### 返回
- **ESP_OK** : Successfully retrieved Wi-Fi disconnect reason
- **ESP_FAIL** : Provisioning app not running

```c
esp_err_t wifi_prov_mgr_configure_sta (wifi_config_t *wifi_cfg)
```

Runs Wi-Fi as Station with the supplied configuration.

Configures the Wi-Fi station mode to connect to the AP with SSID and password specified in config structure and sets Wi-Fi to run as station.

This is automatically called by provisioning service upon receiving new credentials.

If credentials are to be supplied to the manager via a different mode other than through protocomm, then this API needs to be called.

Event WIFI_PROV_CRED_RECV is emitted after credentials have been applied and Wi-Fi station started.

- **wifi_cfg** – [in] Pointer to Wi-Fi configuration structure

### 返回
- **ESP_OK** : Wi-Fi configured and started successfully
- **ESP_FAIL** : Failed to set configuration

```c
esp_err_t wifi_prov_mgr_reset_provisioning (void)
```

Reset Wi-Fi provisioning config.

Calling this API will restore WiFi stack persistent settings to default values.

### 返回
- **ESP_OK** : Reset provisioning config successfully
**Chapter 2. API Reference**

- ESP_FAIL: Failed to reset provisioning config

```c
esp_err_t wifi_prov_mgr_reset_sm_state_on_failure( void )
```

This API should be used to restart provisioning ONLY in the case of provisioning failures without rebooting the device.

- ESP_OK: Reset provisioning state machine successfully
- ESP_FAIL: Failed to reset provisioning state machine
- ESP_ERR_INVALID_STATE: Manager not initialized

```c
esp_err_t wifi_prov_mgr_reset_sm_state_for_reprovision( void )
```

This API can be used to restart provisioning ONLY in case the device is to be provisioned again for new credentials after a previous successful provisioning without rebooting the device.

```c
void user_data
```

User context data to pass as parameter to callback function

**Structures**

```c
struct wifi_prov_event_handler_t
```

Event handler that is used by the manager while provisioning service is active.

**Public Members**

```c
wifi_prov_scheme event_scheme
```

Callback function to be executed on provisioning events

```c
void* user_data
```

User context data to pass as parameter to callback function

**Structures**

```c
struct wifi_prov_scheme
```

Structure for specifying the provisioning scheme to be followed by the manager.

```c
wifi_prov_scheme_ble
wifi_prov_scheme_softap
wifi_prov_scheme_console
```

Ready to use schemes are available:

- wifi_prov_scheme_ble: for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap: for provisioning over SoftAP transport + HTTP server
- wifi_prov_scheme_console: for provisioning over Serial UART transport + Console (for debugging)

**Public Members**
esp_err_t (*prov_start)(protocomm_t *pc, void *config)

Function which is to be called by the manager when it is to start the provisioning service associated with a protocomm instance and a scheme specific configuration

esp_err_t (*prov_stop)(protocomm_t *pc)

Function which is to be called by the manager to stop the provisioning service previously associated with a protocomm instance

void *(*new_config)(void)

Function which is to be called by the manager to generate a new configuration for the provisioning service, that is to be passed to prov_start()

void (*delete_config)(void *config)

Function which is to be called by the manager to delete a configuration generated using new_config()

esp_err_t (*set_config_service)(void *config, const char *service_name, const char *service_key)

Function which is to be called by the manager to set the service name and key values in the configuration structure

esp_err_t (*set_config_endpoint)(void *config, const char *endpoint_name, uint16_t uuid)

Function which is to be called by the manager to set a protocomm endpoint with an identifying name and UUID in the configuration structure

wifi_mode_t wifi_mode

Sets mode of operation of Wi-Fi during provisioning This is set to:

- WIFI_MODE_APSTA for SoftAP transport
- WIFI_MODE_STA for BLE transport

struct wifi_prov_mgr_config_t

Structure for specifying the manager configuration.

Public Members

wifi_prov_scheme_t scheme

Provisioning scheme to use. Following schemes are already available:

- wifi_prov_scheme_ble: for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap: for provisioning over SoftAP transport + HTTP server + mDNS (optional)
- wifi_prov_scheme_console: for provisioning over Serial UART transport + Console (for debugging)

wifi_prov_event_handler_t scheme_event_handler

Event handler required by the scheme for incorporating scheme specific behavior while provisioning manager is running. Various options may be provided by the scheme for setting this field. Use WIFI_PROV_EVENT_HANDLER_NONE when not used. When using scheme wifi_prov_scheme_ble, the following options are available:

- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE
- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT
**wifi_prov_event_handler_t** `app_event_handler`  
Event handler that can be set for the purpose of incorporating application specific behavior. Use WIFI_PROV_EVENT_HANDLER_NONE when not used.

### Macros

**WIFI_PROV_EVENT_HANDLER_NONE**  
Event handler can be set to none if not used.

### Type Definitions

typedef void (*`wifi_prov_cb_func_t`)(void *user_data, `wifi_prov_cb_event_t` event, void *event_data)

typedef struct `wifi_prov_scheme` `wifi_prov_scheme_t`  
Structure for specifying the provisioning scheme to be followed by the manager.

<table>
<thead>
<tr>
<th>备注：</th>
<th>Ready to use schemes are available:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <code>wifi_prov_scheme_ble</code> : for provisioning over BLE transport + GATT server</td>
<td></td>
</tr>
<tr>
<td>- <code>wifi_prov_scheme_softap</code> : for provisioning over SoftAP transport + HTTP server</td>
<td></td>
</tr>
<tr>
<td>- <code>wifi_prov_scheme_console</code> : for provisioning over Serial UART transport + Console (for debugging)</td>
<td></td>
</tr>
</tbody>
</table>

 typedef enum `wifi_prov_security` `wifi_prov_security_t`  
Security modes supported by the Provisioning Manager. These are same as the security modes provided by protocomm

typedef `protocomm_security2_params_t` `wifi_prov_security2_params_t`  
Security 2 params structure This needs to be passed when using WIFI_PROV_SECURITY_2.

### Enumerations

typedef enum `wifi_prov_cb_event_t`  
Events generated by manager. These events are generated in order of declaration and, for the stretch of time between initialization and de-initialization of the manager, each event is signaled only once

<table>
<thead>
<tr>
<th>Values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>-wifi_prov_init</td>
</tr>
<tr>
<td>-WIFI_PROV_START</td>
</tr>
<tr>
<td>-WIFI_PROV_CRED_RECV</td>
</tr>
</tbody>
</table>

enumerator **WIFI_PROV_INIT**  
Emitted when the manager is initialized

enumerator **WIFI_PROV_START**  
Indicates that provisioning has started

enumerator **WIFI_PROV_CRED_RECV**  
Emitted when Wi-Fi AP credentials are received via protocomm endpoint wifi_config. The event data in this case is a pointer to the corresponding `wifi_sta_config_t` structure
enumerator WIFI_PROV_CRED_FAIL
Emitted when device fails to connect to the AP of which the credentials were received earlier on event WIFI_PROV_CRED_RECV. The event data in this case is a pointer to the disconnection reason code with type wifi_prov_sta_fail_reason_t.

enumerator WIFI_PROV_CRED_SUCCESS
Emitted when device successfully connects to the AP of which the credentials were received earlier on event WIFI_PROV_CRED_RECV.

enumerator WIFI_PROV_END
Signals that provisioning service has stopped.

enumerator WIFI_PROV_DEINIT
Signals that manager has been de-initialized.

enum wifi_prov_security
Security modes supported by the Provisioning Manager. These are same as the security modes provided by protocomm.

Values:

enumerator WIFI_PROV_SECURITY_0
No security (plain-text communication).

enumerator WIFI_PROV_SECURITY_1
This secure communication mode consists of X25519 key exchange
- proof of possession (pop) based authentication
- AES-CTR encryption

enumerator WIFI_PROV_SECURITY_2
This secure communication mode consists of SRP6a based authentication and key exchange
- AES-GCM encryption/decryption

Header File

- components/wifi_provisioning/include/wifi_provisioning/scheme_ble.h

Functions

void wifi_prov_scheme_ble_event_cb_free_btdm (void *user_data, wifi_prov_cb_event_t event, void *event_data)

void wifi_prov_scheme_ble_event_cb_free_ble (void *user_data, wifi_prov_cb_event_t event, void *event_data)

void wifi_prov_scheme_ble_event_cb_free_bt (void *user_data, wifi_prov_cb_event_t event, void *event_data)

esp_err_t wifi_prov_scheme_ble_set_service_uuid (uint8_t *uuid128)
Set the 128 bit GATT service UUID used for provisioning.
This API is used to override the default 128 bit provisioning service UUID, which is 0000fff-0000-1000-8000-00805f9b34fb.
This must be called before starting provisioning, i.e., before making a call to wifi_prov_mgr_start_provisioning(), otherwise the default UUID will be used.

备注: The data being pointed to by the argument must be valid at least until provisioning is started. Upon start, the manager will store an internal copy of this UUID, and this data can be freed or invalidated afterwords.

参数 uuid128 - [in] A custom 128 bit UUID

返回
  • ESP_OK : Success
  • ESP_ERR_INVALID_ARG : Null argument

esp_err_t wifi_prov_scheme_ble_set_mfg_data (uint8_t *mfg_data, ssize_t mfg_data_len)

Set manufacturer specific data in scan response.

This must be called before starting provisioning, i.e., before making a call to wifi_prov_mgr_start_provisioning().

备注: It is important to understand that the length of the custom manufacturer data should be within limits. The manufacturer data goes into the scan response along with BLE device name. By default, BLE device name length is of 11 Bytes, however it can vary as per application use case. So, one has to honour the scan response data size limits i.e., \((\text{mfg\_data\_len} + 2) < 31 - (\text{device\_name\_length} + 2)\). If the mfg_data length exceeds this limit, the length will be truncated.

参数
  • mfg_data - [in] Custom manufacturer data
  • mfg_data_len - [in] Manufacturer data length

返回
  • ESP_OK : Success
  • ESP_ERR_INVALID_ARG : Null argument

Macros

WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM

WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE

WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT

Header File

• components/wifi_provisioning/include/wifi_provisioning/scheme_softap.h

Functions

void wifi_prov_scheme_softap_set_httpd_handle (void *handle)

Provide HTTPD Server handle externally.

Useful in cases wherein applications need the webserver for some different operations, and do not want the wifi provisioning component to start/stop a new instance.

备注: This API should be called before wifi_prov_mgr_start_provisioning()
Chapter 2. API

Header File

- components/wifi_provisioning/include/wifi_provisioning/scheme_console.h

- components/wifi_provisioning/include/wifi_provisioning/wifi_config.h

Functions

```c
esp_err_t wifi_prov_config_data_handler (uint32_t session_id, const uint8_t* inbuf, ssize_t inlen,
                                          uint8_t** outbuf, ssize_t *outlen, void *priv_data)
```

Handler for receiving and responding to requests from master.

This is to be registered as the wifi_config endpoint handler (protocomm
protocomm_req_handler_t) using protocomm_add_endpoint()

Structures

```c
struct wifi_prov_sta_conn_info_t
```

WiFi STA connected status information.

**Public Members**

```c
char ip_addr[IP4ADDR_STRLEN_MAX]
```

IP Address received by station

```c
char bssid[6]
```

BSSID of the AP to which connection was established

```c
char ssid[33]
```

SSID of the to which connection was established

```c
uint8_t channel
```

Channel of the AP

```c
uint8_t auth_mode
```

Authorization mode of the AP

```c
struct wifi_prov_config_get_data_t
```

WiFi status data to be sent in response to get_status request from master.

**Public Members**

```c
wifi_prov_sta_state_t wifi_state
```

WiFi state of the station

```c
wifi_prov_sta_fail_reason_t fail_reason
```

Reason for disconnection (valid only when wifi_state is WIFI_STATION_DISCONNECTED)
Chapter 2. API Reference

### wifi_prov_sta_conn_info_t conn_info
Connection information (valid only when wifi_state is WIFI_STATION_CONNECTED)

```c
struct wifi_prov_sta_conn_info_t
{
    // Connection information
    // ...
}
```

### struct wifi_prov_config_set_data_t
WiFi config data received by slave during set_config request from master.

#### Public Members

- `char ssid[33]`  
  SSID of the AP to which the slave is to be connected
- `char password[64]`  
  Password of the AP
- `char bssid[6]`  
  BSSID of the AP
- `uint8_t channel`  
  Channel of the AP

### struct wifi_prov_config_handlers
Internal handlers for receiving and responding to protocomm requests from master.

This is to be passed as priv_data for protocomm request handler (refer to `wifi_prov_config_data_handler()`) when calling `protocomm_add_endpoint()`.

#### Public Members

- `esp_err_t (*get_status_handler)(wifi_prov_config_get_data_t *resp_data, wifi_prov_ctx_t **ctx)`  
  Handler function called when connection status of the slave (in WiFi station mode) is requested
- `esp_err_t (*set_config_handler)(const wifi_prov_config_set_data_t *req_data, wifi_prov_ctx_t **ctx)`  
  Handler function called when WiFi connection configuration (e.g. AP SSID, password, etc.) of the slave (in WiFi station mode) is to be set to user provided values
- `esp_err_t (*apply_config_handler)(wifi_prov_ctx_t **ctx)`  
  Handler function for applying the configuration that was set in set_config_handler. After applying the station may get connected to the AP or may fail to connect. The slave must be ready to convey the updated connection status information when get_status_handler is invoked again by the master.

- `wifi_prov_ctx_t *ctx`  
  Context pointer to be passed to above handler functions upon invocation

### Type Definitions

typedef struct wifi_prov_ctx wifi_prov_ctx_t

Type of context data passed to each get/set/apply handler function set in `wifi_prov_config_handlers` structure.

This is passed as an opaque pointer, thereby allowing it to be defined later in application code as per requirements.
typedef struct wifi_prov_config_handlers wifi_prov_config_handlers_t

Internal handlers for receiving and responding to protocomm requests from master.

This is to be passed as priv_data for protocomm request handler (refer to wifi_prov_config_data_handler()) when calling protocomm_add_endpoint().

**Enumerations**

enum wifi_prov_sta_state_t

WiFi STA status for conveying back to the provisioning master.

Values:

- enumerator WIFI_PROV_STA_CONNECTING
- enumerator WIFI_PROV_STA_CONNECTED
- enumerator WIFI_PROV_STA_DISCONNECTED

enum wifi_prov_sta_fail_reason_t

WiFi STA connection fail reason.

Values:

- enumerator WIFI_PROV_STA_AUTH_ERROR
- enumerator WIFI_PROV_STA_AP_NOT_FOUND

本部分的 API 示例代码存放在 ESP-IDF 示例项目的 provisioning 目录下。

2.9 存储 API

2.9.1 FAT 文件系统

ESP-IDF 使用 FatFs 库来实现 FAT 文件系统。FatFs 库位于 fatfs 组件中，您可以直接使用，也可以借助 C 标准库和 POSIX API 通过 VFS（虚拟文件系统）使用 FatFs 库的大多数功能。

此外，我们对 FatFs 库进行了扩展，新增了支持可插拔磁盘 I/O 调度层，从而允许在运行时将 FatFs 驱动映射到物理磁盘。

FatFs 与 VFS 配合使用

头文件 fatfs/vfs/esp_vfs_fat.h 定义了连接 FatFs 和 VFS 的函数。

函数 esp_vfs_fat_register() 分配一个 FATFS 结构，并在 VFS 中注册特定路径前缀。如果文件路径以此前缀开头，则对此文件的后续操作将转至 FatFs API。
函数 `esp_vfs_fat_unregister_path()` 删除在 VFS 中的注册，并释放 FATFS 结构。多数应用程序在使用 `esp_vfs_fat_` 函数时，采用如下步骤：

1. **调用 `esp_vfs_fat_register()`，指定**：
   - 挂载文件系统的路径前缀（例如，“/sdcard”或“/spiflash”）
   - FatFs 驱动编号
   - 一个用于接收指向 FATFS 结构指针的变量
2. **调用 `ff_diskio_register()`**，为步骤 1 中的驱动编号注册磁盘 I/O 驱动；
3. **调用 FatFs 函数 `f_mount`，随后调用 `f_diskio` 或 `f_mkfs`，并使用与传递到 `esp_vfs_fat_register()` 相同的驱动编号挂载文件系统。请参考 FatFs 文档，查看更多信息；
4. **调用 C 标准库和 POSIX API 对路径中带有步骤 1 中所述前缀的文件（例如，“/sdcard/hello.txt”）执行打开、读取、写入、擦除、复制等操作。文件系统默认使用 8.3 文件名格式 (SFN)。若您需要使用长文件名 (LFN)，启用 `CONFIG_FATFS_LONG_FILENAMES` 选项。请参考这里，查看更多信息；
5. **您可以选择启用 `CONFIG_FATFS_USE_FASTSEEK` 选项，使用 POSIX lseek 来快速执行。快速查找不适用于编辑模式下的文件，所以，使用快速查找时，应在只读模式下打开（或者关闭然后重新打开）文件；
6. **您也可以选择直接调用 FatFs 库函数，但需要使用没有 VFS 前缀的路径（例如，“/hello.txt”）
7. 关闭所有打开的文件；
8. **调用 FatFs 函数 `f_mount` 并使用 NULL FATFS* 参数，为与上述编号相同的驱动卸载文件系统；
9. **调用 FatFs 函数 `ff_diskio_register()` 并使用 NULL `ff_diskio_impl_t*` 参数和相同的驱动编号，来释放注册的磁盘 I/O 驱动；
10. **调用 `esp_vfs_fat_unregister_path()` 并使用文件系统挂载的路径将 FatFs 从 VFS 中移除，并释放步骤 1 中分配的 FATFS 结构。

便捷函数 `esp_vfs_fat_sdmmc_mount`、`esp_vfs_fat_sdspi_mount` 和 `esp_vfs_fat_sdmmcUnmount` 对上述步骤进行了封装，并加入了对 SD 卡初始化的处理。我们将在下一章节详细介绍以上函数。

```c
esp_err_t esp_vfs_fat_register(const char* base_path, const char* fat_drive, size_t max_files, FATFS* out_fs)
```

Register FatFs with VFS component.

This function registers given FAT drive in VFS, at the specified base path. If only one drive is used, fat_drive argument can be an empty string. Refer to FATFS library documentation on how to specify FAT drive. This function also allocates FATFS structure which should be used for f_mount call.

**备注:** This function doesn’t mount the drive into FATFS, it just connects POSIX and C standard library IO function with FATFS. You need to mount desired drive into FATFS separately.

**参数**

- `base_path` – path prefix where FATFS should be registered
- `fat_drive` – FATFS drive specification; if only one drive is used, can be an empty string
- `max_files` – maximum number of files which can be open at the same time
- `out_fs` – [out] pointer to FATFS structure which can be used for FATFS f_mount call is returned via this argument.

**返回**

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_register was already called
- ESP_ERR_NO_MEM if not enough memory or too many VFSes already registered

```c
esp_err_t esp_vfs_fat_unregister_path(const char* base_path)
```

Un-register FatFs from VFS.

**备注:** FATFS structure returned by esp_vfs_fat_register is destroyed after this call. Make sure to call f_mount function to unmount it before calling esp_vfs_fat_unregister_ctx. Difference between this function and the one
above is that this one will release the correct drive, while the one above will release the last registered one

参数 **base_path** — path prefix where FATFS is registered. This is the same used when esp_vfs_fat_register was called

返回
- ESP_OK on success
- ESP_ERR_INVALID_STATE if FATFS is not registered in VFS

### FatFs 与 VFS 和 SD 卡配合使用

头文件 fatfs/vfs/esp_vfs_fat.h 定义了便捷函数 esp_vfs_fat_sdmmc_mount()，esp_vfs_fat_sdspi_mount() 和 esp_vfs_fat_sdcard_unmount()。这些函数分别执行上一章节的步骤 1-3 和步骤 7-9，并初始化 SD 卡，但仅提供有限的错误处理功能。我们鼓励开发人员查看源代码，将更多高级功能集成到产品应用中。

便捷函数 esp_vfs_fat_sdmmc_mount() 用于卸载文件系统并释放从 esp_vfs_fat_sdmmc_mount() 函数获取的资源。

```c
esp_err_t esp_vfs_fat_sdmmc_mount(const char *base_path, const sdmmc_host_t *host_config, const void *slot_config, const esp_vfs_fat_mount_config_t *mount_config, sdmmc_card_t **out_card)
```

Convenience function to get FAT filesystem on SD card registered in VFS.

This is an all-in-one function which does the following:

- initializes SDMMC driver or SPI driver with configuration in host_config
- initializes SD card with configuration in slot_config
- mounts FAT partition on SD card using FATFS library, with configuration in mount_config
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact. For real world applications, developers should implement the logic of probing SD card, locating and mounting partition, and registering FATFS in VFS, with proper error checking and handling of exceptional conditions.

备注: Use this API to mount a card through SDSPI is deprecated. Please call esp_vfs_fat_sdspi_mount() instead for that case.

参数
- **base_path** — path where partition should be registered (e.g. “/sdcard”)
- **host_config** — Pointer to structure describing SDMMC host. When using SDMMC peripheral, this structure can be initialized using SDMMC_HOST_DEFAULT() macro. When using SPI peripheral, this structure can be initialized using SD_SI_HOST_DEFAULT() macro.
- **slot_config** — Pointer to structure with slot configuration. For SDMMC peripheral, pass a pointer to sdmmc_slot_config_t structure initialized using SDMMC_SLOT_CONFIG_DEFAULT.
- **mount_config** — pointer to structure with extra parameters for mounting FATFS
- **out_card** [out] if not NULL, pointer to the card information structure will be returned via this argument

返回
- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount was already called
- ESP_ERR_NO_MEM if memory cannot be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from SDMMC or SPI drivers, SDMMC protocol, or FATFS drivers
**esp_err_t esp_vfs_fat_sdmmc_unmount**(void)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_sdmmc_mount.

**Deprecated:**
Use esp_vfs_fat_sdcard_unmount() instead.

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount hasn’t been called

**esp_err_t esp_vfs_fat_sdspi_mount**(const char*base_path, const sdmmc_host_t*host_config_input, const sdspi_device_config_t*slot_config, const esp_vfs_fat_mount_config_t*mount_config, sdmmc_card_t**out_card)

Convenience function to get FAT filesystem on SD card registered in VFS.

This is an all-in-one function which does the following:

- initializes an SPI Master device based on the SPI Master driver with configuration in slot_config, and attach it to an initialized SPI bus.
- initializes SD card with configuration in host_config_input
- mounts FAT partition on SD card using FATFS library, with configuration in mount_config
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact. For real world applications, developers should implement the logic of probing SD card, locating and mounting partition, and registering FATFS in VFS, with proper error checking and handling of exceptional conditions.

**备注:** This function try to attach the new SD SPI device to the bus specified in host_config. Make sure the SPI bus specified in host_config->slot have been initialized by spi_bus_initialize() before.

**参数**

- **base_path** – path where partition should be registered (e.g. “/sdcard”)
- **host_config_input** – Pointer to structure describing SDMMC host. This structure can be initialized using SDSPI_HOST_DEFAULT() macro.
- **slot_config** – Pointer to structure with slot configuration. For SPI peripheral, pass a pointer to sdspi_device_config_t structure initialized using SDSPI_DEVICE_CONFIG_DEFAULT().
- **mount_config** – pointer to structure with extra parameters for mounting FATFS
- **out_card** [out] If not NULL, pointer to the card information structure will be returned via this argument. It is suggested to hold this handle and use it to unmount the card later if needed. Otherwise it’s not suggested to use more than one card at the same time and unmount one of them in your application.

**返回**

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount was already called
- ESP_ERR_NO_MEM if memory cannot be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from SDMMC or SPI drivers, SDMMC protocol, or FATFS drivers

**struct esp_vfs_fat_mount_config_t**

Configuration arguments for esp_vfs_fat_sdmmc_mount and esp_vfs_fat_spiflash_mount_rw_wl functions.

**Public Members**
bool `format_if_mount_failed`
If FAT partition can not be mounted, and this parameter is true, create partition table and format the filesystem.

int `max_files`
Max number of open files.

`size_t allocation_unit_size`
If format_if_mount_failed is set, and mount fails, format the card with given allocation unit size. Must be a power of 2, between sector size and 128 * sector size. For SD cards, sector size is always 512 bytes. For wear levelling, sector size is determined by `CONFIG_WL_SECTOR_SIZE` option.

Using larger allocation unit size will result in higher read/write performance and higher overhead when storing small files.

Setting this field to 0 will result in allocation unit set to the sector size.

bool `disk_status_check_enable`
Enables real `ff_disk_status` function implementation for SD cards (ff_sdmmc_status). Possibly slows down IO performance.

Try to enable if you need to handle situations when SD cards are not unmounted properly before physical removal or you are experiencing issues with SD cards.

Doesn’t do anything for other memory storage media.

`esp_err_t esp_vfs_fat_sdcardUnmount` (const char* `base_path`, `sdmmc_card_t* card`)
Unmount an SD card from the FAT filesystem and release resources acquired using `esp_vfs_fat_sdmmc_mount()` or `esp_vfs_fat_sdspi_mount()`

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the card argument is unregistered
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount hasn’t been called

FatFs 与 VFS 配合使用（只读模式下）

头文件 `fatfs/vfs/esp_vfs_fat.h` 也定义了两个便捷函数 `esp_vfs_fat_spiflash_mount_ro()` 和 `esp_vfs_fat_spiflashUnmount_ro()`。上述两个函数分别对 FAT 只读分区执行步骤 1-3 和步骤 7-9。有些数据分区仅在工厂配置时写入一次，之后在整个硬件生命周期内都不会再有任何改动。利用上述两个函数处理这种数据分区非常方便。

`esp_err_t esp_vfs_fat_spiflashMount_ro` (const char* `base_path`, const char* `partition_label`, const 
`esp_vfs_fat_mount_config_t* mount_config`)

Convenience function to initialize read-only FAT filesystem and register it in VFS.

This is an all-in-one function which does the following:

- finds the partition with defined partition_label. Partition label should be configured in the partition table.
- mounts FAT partition using FATFS library
- registers FATFS library with VFS, with prefix given by base_prefix variable

备注：Wear levelling is not used when FAT is mounted in read-only mode using this function.

参数
- base_path – path where FATFS partition should be mounted (e.g. “/spiflash”)
• **partition_label** – label of the partition which should be used
• **mount_config** – pointer to structure with extra parameters for mounting FATFS

返回
• ESP_OK on success
• ESP_ERR_NOT_FOUND if the partition table does not contain FATFS partition with given label
• ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_ro was already called for the same partition
• ESP_ERR_NO_MEM if memory cannot be allocated
• ESP_FAIL if partition cannot be mounted
• other error codes from SPI flash driver, or FATFS drivers

```c
esp_err_t esp_vfs_fat_spiflash_unmount_ro(const char*base_path, const char*partition_label)
```

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_spiflash_mount_ro.

参数
• **base_path** – path where partition should be registered (e.g. “/spiflash”)
• **partition_label** – label of partition to be unmounted

返回
• ESP_OK on success
• ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_ro has not been called

FatFs 磁盘 I/O 层

我们对 FatFs API 函数进行了扩展，实现了运行期间注册磁盘 I/O 驱动。

上述 API 为 SD/MMC 卡提供了磁盘 I/O 函数实现方式，可使用 `ff_diskio_register_sdmmc()` 函数注册指定的 FatFs 驱动编号。

```c
void ff_diskio_register(BYTE pdrv, const ff_diskio_impl_t *discio_impl)
```

Register or unregister disk io driver for given drive number.

When FATFS library calls one of disk_xxx functions for driver number pdrv, corresponding function in discio_impl for given pdrv will be called.

参数
• **pdrv** – drive number
• **discio_impl** – pointer to `ff_diskio_impl_t` structure with diskio functions or NULL to unregister and free previously registered drive

```c
struct ff_diskio_impl_t
```

Structure of pointers to disk IO driver functions.

See FatFs documentation for details about these functions

**Public Members**

DSTATUS (*init)(unsigned char pdrv)

disk initialization function

```c
DSTATUS (*status)(unsigned char pdrv)
```

disk status check function

```c
DRESULT (*read)(unsigned char pdrv, unsigned char *buff, uint32_t sector, unsigned count)
```

sector read function
### Chapter 2. API Reference

**DRESULT** (*write*) (unsigned char pdrv, const unsigned char *buff, uint32_t sector, unsigned count)

sector write function

**DRESULT** (*ioctl*) (unsigned char pdrv, unsigned char cmd, void *buff)

function to get info about disk and do some misc operations

void **ff_diskio_register_sdmmc** (unsigned char pdrv, sdmmc_card_t *card)

Register SD/MMC disk io driver

- **pdrv** - drive number
- **card** - pointer to *sdmmc_card_t* structure describing a card; card should be initialized before calling _f_mount_.

**esp_err_t** **ff_diskio_register_wl_partition** (unsigned char pdrv, wl_handle_t flash_handle)

Register spi flash partition

- **pdrv** - drive number
- **flash_handle** - handle of the wear levelling partition.

**esp_err_t** **ff_diskio_register_raw_partition** (unsigned char pdrv, const esp_partition_t *part_handle)

Register spi flash partition

- **pdrv** - drive number
- **part_handle** - pointer to raw flash partition.

### FatFs 分区生成器

我们为 FatFs (wl_fatfsgen.py) 提供了分区生成器，该生成器集成在构建系统中，方便用户在自己的项目中使用。

该生成器可以在主上创建文件系统镜像，并用指定的主机文件夹内容对其进行填充。

该脚本是建立在分区生成器的基础（fatfsgen.py），目前除了可以生成分区外，也可以初始化磨损均衡。

目前的最新版本支持短文件名，长文件名、FAT12 和 FAT16。长文件名的上限是 255 个字符，文件名中可以包含多个 . 符号以及其他字符，如 +, /, ;, =, [ and ] 等。

#### 构建系统中使用 FatFs 分区生成器

通过调用 fatfs_create_partition_image 可以直接从 CMake 构建系统中调用 FatFs 分区生成器:

```bash
fatfs_create_spiflash_image(<partition> <base_dir> [FLASH_IN_PROJECT])
```

如果不希望在生成分区时使用磨损均衡，可以使用 fatfs_create_rawflash_image:

```bash
fatfs_create_rawflash_image(<partition> <base_dir> [FLASH_IN_PROJECT])
```

fatfs_create_spiflash_image 以及 fatfs_create_rawflash_image 必须从项目的 CMake-Lists.txt 中调用。

如果您决定使用 fatfs_create_rawflash_image（不支持磨损均衡），请注意它仅支持在设备中以只读模式安装。

该函数的参数如下:

1. **partition** - 分区的名称，需要在分区表中定义（如 storage/fatfsgen/partitions_example.csv）。
2. **base_dir** - 目录名称，该目录会被编码为 FatFs 分区，也可以选择将其被烧录进设备。但注意必须在分区表中指定合适的分区大小。
3. FLASH_IN_PROJECT 标志 - 可选参数，用户可以通过指定 FLASH_IN_PROJECT，选择在执行 idf.py flash -p <PORT> 时让分区镜像自动与应用程序二进制文件、分区表等一同烧录进设备。

4. PRESERVE_TIME 标志 - 可选参数，用户可强制让目标镜像保留源文件夹的时间戳。如果不保留，每个目标镜像的时间戳都将设置为 FATFS 默认初始时间（1980 年 1 月 1 日）。

例如:

```python
fatfs_create_partition_image(my_fatfs_partition my_folder FLASH_IN_PROJECT)
```

没有指定 FLASH_IN_PROJECT 时也可以生成分区镜像，但是用户需要使用 esptool.py 或自定义的构建系统目标对其手动烧录。

相关示例请查看 storage/fatfsgen。

**FatFs 分区分析器**

我们为 FatFs 提供分区分析器 (fatfsparse.py)。

该分析器为 FatFs 分区生成器 (fatfsgen.py) 的逆向工具，可以根据 FatFs 镜像在主机上生成文件夹结构。

您可以使用:

```
./fatfsparse.py [-h] [--wl-layer {detect,enabled,disabled}] fatfs_image.img
```

### 2.9.2 量产程序

**介绍**

这一程序主要用于量产时为每一设备创建工厂 NVS（非易失性存储器）分区镜像。NVS 分区镜像由 CSV（逗号分隔值）文件生成，文件中包含了用户提供的配置项及配置值。

注意，该程序仅创建用于量产的二进制镜像，您需要使用以下工具将镜像烧录到设备上：

- esptool.py
- Flash 下载工具（仅适用于 Windows）。下载后解压，然后按照 doc 文件夹中的说明操作。
- 使用定制的生产工具直接烧录程序

**准备工作**

该程序依赖于 esp-idf 的 NVS 分区程序

- 操作系统要求:
  - Linux、MacOS 或 Windows（标准版）
- 安装依赖包:
  - Python

**备注:**

使用该程序之前，请确保:

- Python 路径已添加到 PATH 环境变量中；
- 已经安装 requirement.txt 中的软件包，requirement.txt 在 esp-idf 根目录下。
具体流程

CSV 配置文件

CSV 配置文件中包含设备待烧录的配置信息，定义了待烧录的配置项。
配置文件中数据格式如下（REPEAT 标签可选）：

```plaintext
name1,namespace,   <!-- 第一个条目应该为 "namespace" 类型
key1,type1,encoding1
key2,type2,encoding2,REPEAT
name2,namespace,
key3,type3,encoding3
key4,type4,encoding4
```

备注：文件第一行应始终为 namespace 条目。

每行应包含三个参数：key, type 和 encoding，并以逗号分隔。如果有 REPEAT 标签，则主 CSV 文件中所有设备此键值均相同。
有关各个参数的详细说明，请参阅 NVS 分区生成程序的 README 文件。

CSV 配置文件示例如下：

```plaintext
app,namespace,
firmware_key,data,hex2bin
serial_no,data,string,REPEAT
device_no,data,i32
```

备注：
请确保：
- 逗号 ‘,’ 前后无空格；
- CSV 文件每行末尾无空格。

主 CSV 文件

主 CSV 文件中包含设备待烧录的详细信息，文件中每行对应一个设备实体。

**主 CSV 文件的数据格式如下：**

```plaintext
key1,key2,key3,....
value1,value2,value3,....
```

备注：文件中键 (key) 名应始终置于文件首行。从配置文件中获取的键，在此文件中的排列顺序应与其在配置文件中的排列顺序相同。主 CSV 文件同时可以包含其它列（键），这些列将被视为元数据，而不会编译进最终二进制文件。
每行应包含相应键的键值 (value)，并用逗号隔开。如果某键带有 REPEAT 标签，则仅需在第二行（即第一个条目）输入对应的值，后面其他行为空。

参数描述如下：

**value**  Data value

value 是与键对应的键值。

主 CSV 文件示例如下：

```console
id,firmware_key,serial_no,device_no
1,1a2b3c4d5e6faabb,A1,101
2,1a2b3c4d5e6fccdd,,102
3,1a2b3c4d5e6feeff,,103
```

**备注：** 如果出现 REPEAT 标签，则会在相同目录下生成一个新的主 CSV 文件用作主输入文件，并在每行为带有 REPEAT 标签的键插入键值。

量产程序还会创建中间 CSV 文件，NVS 分区程序将使用此 CSV 文件作为输入，然后生成二进制文件。

中间 CSV 文件的格式如下：

```console
key,type,encoding,value
key,namespace, ,
key1,type1,encoding1,value1
key2,type2,encoding2,value2
```

此步骤将为每一设备生成一个中间 CSV 文件。

**运行量产程序**

**使用方法：**

```
python mfg_gen.py [-h] {generate,generate-key} ...
```

**可选参数：**

<table>
<thead>
<tr>
<th>序号</th>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-h,--help</td>
<td>显示帮助信息并退出</td>
</tr>
</tbody>
</table>

**命令：**

运行 mfg_gen.py [command] -h 查看更多帮助信息

<table>
<thead>
<tr>
<th>序号</th>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>generate</td>
<td>生成 NVS 分区</td>
</tr>
<tr>
<td>2</td>
<td>generate-key</td>
<td>生成加密密钥</td>
</tr>
</tbody>
</table>

为每个设备生成工厂镜像 (默认)

**使用方法：**

```
python mfg_gen.py generate [-h] [--fileid FILEID] [--version {1,2}] [--keygen]
    [--keyfile KEYFILE] [--inputkey INPUTKEY]
    [--outdir OUTDIR]
    conf values prefix size
```
位置参数:

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf</td>
<td>待解析的 CSV 配置文件路径</td>
</tr>
<tr>
<td>values</td>
<td>待解析的主 CSV 文件路径</td>
</tr>
<tr>
<td>prefix</td>
<td>每个输出文件名前缀的唯一名称</td>
</tr>
<tr>
<td>size</td>
<td>NVS 分区大小（以字节为单位，且为 4096 的整数倍）</td>
</tr>
</tbody>
</table>

可选参数:

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>显示帮助信息并退出</td>
</tr>
<tr>
<td>--fileid FILEID</td>
<td>每个文件名后缀的唯一值标识符（主 CSV 文件中的任意键），默认为数值 1, 2, 3...</td>
</tr>
<tr>
<td>--version [1,2]</td>
<td>- 设定多页 Blob 版本。  - 版本 1 - 禁用多页 Blob。  - 版本 2 - 启用多页 Blob。  - 默认版本：版本 2</td>
</tr>
<tr>
<td>--keygen</td>
<td>生成 NVS 分区加密密钥</td>
</tr>
<tr>
<td>--inputkey INPUTKEY</td>
<td>内含 NVS 分区加密密钥的文件</td>
</tr>
<tr>
<td>--outdir OUTDIR</td>
<td>输出目录，用于存储创建的文件（默认当前目录）</td>
</tr>
</tbody>
</table>

请运行以下命令为每个设备生成工厂镜像，量产程序同时提供了一个 CSV 示例文件:

```bash
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_--singlepage_blob.csv Sample 0x3000
```

主 CSV 文件应在 file 类型下设置一个相对路径，相对于运行该程序的当前目录。

为每个设备生成工厂加密镜像

运行以下命令为每个设备生成工厂加密镜像，量产程序同时提供了一个 CSV 示例文件。

- 通过量产程序生成加密密钥来进行加密:
  ```bash
  python mfg_gen.py generate samples/sample_config.csv samples/sample_values_--singlepage_blob.csv Sample 0x3000 --keygen
  ```

备注：创建的加密密钥格式为 <outdir>/keys/keys-<prefix>-<fileid>.bin。加密密钥存储于新建文件的 keys/ 目录下，与 NVS 密钥分区兼容。更多信息请参考 NVS 密钥分区。

- 提供加密密钥用作二进制输入文件来进行加密:
  ```bash
  python mfg_gen.py generate samples/sample_config.csv samples/sample_values_--singlepage_blob.csv Sample 0x3000 --inputkey keys/sample_keys.bin
  ```

仅生成加密密钥

使用方法:

```bash
python mfg_gen.py generate-key [-h] [--keyfile KEYFILE] [--outdir OUTDIR]
```

可选参数：+ | -h, --help | 显示帮助信息并退出 |
| +------------------|------------------|
| | --keyfile KEYFILE | 加密密钥文件的输出路径 |
| | --outdir OUTDIR | 输出目录，用于存储创建的文件（默认当前目录） |

Submit Document Feedback
运行以下命令仅生成加密密钥:

```
python mfg_gen.py generate-key
```

**备注：** 创建的加密密钥格式为 `<outdir>/keys/keys-<timestamp>.bin`。时间戳格式为：
%m-%d_%H-%M。如需自定义目标文件名，请使用 `keyfile` 参数。

生成的加密密钥二进制文件还可以用于为每个设备的工厂镜像加密。

`fileid` 参数的默认值为 1, 2, 3…，与 CSV 文件中的行一一对应，内含设备配置值。

运行量产程序时，将在指定的 `outdir` 目录下创建以下文件夹:

* `bin/` 存储生成的二进制文件
* `csv/` 存储生成的中间 CSV 文件
* `keys/` 存储加密密钥（创建工厂加密镜像时会用到）

### 2.9.3 非易失性存储库

#### 简介

非易失性存储 (NVS) 库主要用于在 flash 中存储键值格式的数据。本文档将详细介绍 NVS 常用的一些概念。

**底层存储** NVS 库通过调用 `esp_partition` API 使用主 flash 的部分空间，即类型为 data 且子类型为 `nvs` 的所有分区。应用程序可调用 `nvs_open()` API 选择使用带有 `nvs` 标签的分区，也可以通过调用 `nvs_open_from_partition()` API 选择使用指定名称的任意分区。

NVS 库后续版本可能会增加其他存储器后端，来将数据保存至其他 flash 芯片（SPI 或 I2C 接口）、RTC 或 FRAM 中。

**备注：** 如果 NVS 分区被截断（例如，更改分区表布局时），则应删除分区内容。可以使用 ESP-IDF 构建系统中的 `idf.py erase-flash` 命令擦除 flash 上的所有内容。

**备注：** NVS 最适合存储一些较小的数据，而非字符串或二进制大对象 (BLOB) 等较大的数据。如需存储较大的 BLOB 或者字符串，请考虑使用基于磨损均衡的 FAT 文件系统。

**键值对** NVS 的操作对象为键值对，其中键是 ASCII 字符串，当前支持的最大键长为 15 个字符。值可以为以下几种类型:

* 整数型：`uint8_t`、`int8_t`、`uint16_t`、`int16_t`、`uint32_t`、`int32_t`、`uint64_t` 和 `int64_t`；
* 以 `0` 结尾的字符串；
* 可变长度的二进制数据 (BLOB)

**备注：** 字符串值当前上限为 4000 字节，其中包括空终止符。BLOB 值上限为 508,000 字节或分区大小的 97.6% 减去 4000 字节，以较低值为准。

后续可能会增加对 float 和 double 等其他类型数据的支持。

键必须唯一，为现有的键写入新的值可能产生如下结果：
Chapter 2. API 参考

- 如果新旧值数据类型相同，则更新值；
- 如果新旧值数据类型不同，则返回错误。

读取时也会执行数据类型检查。如果读取操作的数据类型与该值的数据类型不匹配，则返回错误。

命名空间 为了减少不同组件之间键名的潜在冲突，NVIS 将每个键值对分配给一个命名空间。命名空间的命名规则遵循键名的命名规则，例如，最多可占 15 个字符。此外，单个 NVIS 分区最多只能容纳 254 个不同的命名空间。命名空间的名称在调用 nvs_open() 或 nvs_open_from_partition() 中指定，调用后将返回一个不透明句柄，用于后续调用 nvs_get_*、nvs_set_* 和 nvs_commit 函数。这样，一个句柄关联一个命名空间。键名便不会与其他命名空间中相同键名冲突。请注意，不同 NVIS 分区中具有相同名称的命名空间将被视为不同的命名空间。

NVIS 迭代器 迭代器允许根据指定的分区名称、命名空间和数据类型轮询 NVIS 中存储的键值对。

您可以使用以下函数，执行相关操作：
- nvs_entry_find: 创建一个不透明句柄，用于后续调用 nvs_entry_next 和 nvs_entry_info 函数；
- nvs_entry_next：让迭代器指向下一个键值对；
- nvs_entry_info：返回每个键值对的信息。

总的来说，所有通过 nvs_entry_find() 获得的迭代器（包括 NULL 迭代器）都必须使用 nvs_release_iterator() 释放。一般情况下，nvs_entry_find() 和 nvs_entry_next() 会将给定的迭代器设置为 NULL 或为一个有效的迭代器。但如果出现参数错误（如返回 ESP_ERR_NVS_NOT_FOUND），给定的迭代器不会被修改。因此，在调用 nvs_entry_find() 之前最好将迭代器初始化为 NULL，这样可以避免在释放迭代器之前进行复杂的错误检查。

安全性、篡改性及鲁棒性 NVIS 与 ESP32 flash 加密系统不直接兼容。但如果 NVIS 加密与 ESP32 flash 加密一起使用时，数据仍可以加密形式存储。详情请参阅 NVIS 加密。

如果未启用 NVIS 加密，任何对 flash 芯片有物理访问权限的用户都可以修改、擦除或添加键值对。NVIS 加密启用后，如果不知道相应的 NVIS 加密密钥，则无法修改或添加键值对并将其识别为有效键值对。但是，针对擦除操作没有相应的防篡改功能。

当 flash 处于不一致状态时，NVIS 库会尝试恢复。在任何时候点关闭设备电源，然后重新打开电源，不会导致数据丢失；但如果关闭设备电源时正在写入新的键值对，这一键值对可能会丢失。该库应该能够在 flash 中存在任何随机数据的情况下正常初始化。

NVIS 加密 NVIS 分区内存储的数据可使用 AES-XTS 进行加密，类似于 IEEE P1619 磁盘加密标准中提到的加密方式。为了实现加密，每个条目被均视为一个扇区，并将条目相对地址（相对于分区开头）传送加密算法，用作扇区号。可通过 CONFIG_NVS_ENCRYPTION 启用 NVIS 加密。NVIS 加密所需的密钥存储于其他分区，并且被 flash 保护。因此，在使用 NVIS 加密前应先启用 flash 保护。

启用 flash 保护时，默认启用 NVIS 加密。这是因为 Wi-Fi 驱动在默认的 NVIS 分区中存储了凭证（如 SSID 和密码）。如已启用平台级加密，那么同时默认启用 NVIS 加密有其必要性。

使用 NVIS 加密，分区表必须包含 NVIS 密钥分区。在分区表选项 (menuconfig > Partition Table) 下，为 NVIS 加密提供了两个包含 NVIS 密钥分区的分区表。您可以通过工程配置菜单 (idf.py menuconfig) 进行选择。请参考 security/flash_encryption 中的例子，了解如何配置和使用 NVIS 加密功能。

NVIS 密钥分区 应用程序如果想使用 NVIS 加密，则需要编译进一个类型为 data，子类型为 key 的密钥分区。该分区应标记为 key 密且最小为 4096 字节。如需了解更多信息，请参考分区表。在分区表选项 (menuconfig > Partition Table) 下提供了两个包含 NVIS 密钥分区的额外分区表，可以直接用于 NVIS 加密，这些分区的具体结构见下表：
### Chapter 2. API 参考

| +---------------------------------------------+ | 1. XTS encryption key (32) | |
| +---------------------------------------------+ | XTS tweak key (32) | |
| +---------------------------------------------+ | CRC32 (4) | |

可以通过以下两种方式生成NVS 密钥分区 中的 XTS 加密密钥：

1. 在 ESP 芯片上生成密钥：
   启用 NVS 密钥时，可用 nvs_flash_init() API 函数来初始化加密的默认 NVS 分区，在内部生成 ESP 芯片上的 XTS 加密密钥。在找到 NVS 密钥分区后，API 函数利用 nvs_flash/include/nvs_flash.h 提供的 nvs_flash_generate_keys() 函数，自动生成并存储该分区中的 NVS 密钥。当各自的密钥分区为空时，才会生成并存储新的密钥。可以借助 nvs_flash_secure_init_partition() 用同一密钥分区来读取安全配置，以初始化一个定制的加密 NVS 分区。

   API 函数 nvs_flash_secure_init() 和 nvs_flash_secure_init_partition() 不在内部产生密钥。当这些 API 函数用于初始化加密的 NVS 分区时，可以在启动后使用 nvs_flash.h 提供的 nvs_flash_generate_keys() API 函数生成密钥，以加密的形式把密钥写入密钥分区内。

   **备注**：请注意，使用该方法启动应用前，必须先完全解除 nvs_keys 分区，否则该应用可能会认为 nvs_keys 分区不为空，并且包含数据格式错误。从而导致 ESP_ERR_NVS_CORRUPT_KEY_PART 报错。如果遇到这种情况，可以使用以下命令：

   ```bash
   parttool.py --port PORT --partition-table-file=PARTITION_TABLE_FILE --
   --partition-table-offset PARTITION_TABLE_OFFSET erase_partition --
   --partition-type=data --partition-subtype=nvs_keys
   ```

2. 使用预先生成的密钥分区：
   若 NVS 密钥分区 中的密钥不是由应用程序生成，则需要使用预先生成的密钥分区。可以使用 NVS 分区生成工具生成包含 XTS 加密密钥的 NVS 密钥分区。用户可以借助以下两个命令，将预先生成的密钥分区储存于 flash 上：

   i) 建立并烧录分区表

   ```bash
   idf.py partition-table partition-table-flash
   ```

   ii) 调用 parttool.py，将密钥存储在 flash 上的 NVS 密钥分区 中。详见 doc:`/api-guides/partition-tables` 的分区工具部分。

   ```bash
   parttool.py --port PORT --partition-table-offset PARTITION_TABLE_-
   --OFFSET write_partition --partition-name="name of nvs_key_partition" -
   --input NVS_KEY_PARTITION_FILE
   ```

   **备注**：如需在设备处于 flash 加密开发模式时更新 NVS 密钥分区，请调用 parttool.py 对 NVS 密钥分区进行加密。同时，由于设备上的分区表也已加密，您还需要在构建目录 (build/partition_table) 中提供一个指向未加密分区表的指针。您可以使用如下命令：

   ```bash
   parttool.py --esptool-write-args encrypt --port PORT --partition-table-
   --file=PARTITION_TABLE_FILE --partition-table-offset PARTITION_TABLE_-
   --OFFSET write_partition --partition-name="name of nvs_key_partition" -
   --input NVS_KEY_PARTITION_FILE
   ```

由于分区已标记为已加密，而且启用了 Flash 加密，引导程序在首次启动时将使用 flash 加密对密钥分区进行加密。

应用程序可以使用不同的密钥对不同的 NVS 分区进行加密，这样就会需要多个加密密钥分区。应用程序应为解密操作提供正确的密钥或密钥分区。
加密读取/写入 nvs_get_* 和 nvs_set_* 等 NVS API 函数同样可以对 NVS 加密分区执行读写操作。

加密默认的 NVS 分区：无需额外步骤即可启用默认 NVS 分区的加密。启用 CONFIG_NVS_ENCRYPTION 时，nvs_flash_init() API 函数会在内部使用找到的第一个 NVS 密钥分区执行额外步骤，以启用默认 NVS 分区的加密（详情请参考 API 文档）。另外，nvs_flash_secure_init() API 函数也可以用来启用默认 NVS 分区的加密。

加密一个自定义的 NVS 分区：使用 nvs_flash_secure_init_partition() API 函数启用自定义 NVS 分区的加密。若非 nvs_flash_init_partition()。

使用 nvs_flash_secure_init() 和 nvs_flash_secure_init_partition() API 函数时，应用程序如需在加密状态下执行 NVS 读写操作，应遵循以下步骤：
1. 使用 esp_partition_find* API 查找密钥分区和 NVS 数据分区；
2. 使用 nvs_flash_read_security_cfg 或 nvs_flash_generate_keys API 填充 nvs_sec_cfg_t 结构；
3. 使用 nvs_flash_secure_init 或 nvs_flash_secure_init_partition API 初始化 NVS flash 分区；
4. 使用 nvs_open 或 nvs_open_from_partition API 打开命名空间；
5. 使用 nvs_get_* 或 nvs_set_* API 执行 NVS 读取/写入操作；
6. 使用 nvs_flash_deinit API 释放已初始化的 NVS 分区。

NVS 分区生成程序

NVS 分区生成程序帮助生成 NVS 分区二进制文件，可使用烧录程序将二进制文件单独烧录至特定分区。烧录至分区上的键值对由 CSV 文件提供，详情请参考 NVS 分区生成程序。

应用示例

ESP-IDF storage 目录下提供了数个代码示例：

storage/nvs_rw_value

演示如何读取及写入 NVS 单个整数值。

此示例中的值表示 ESP32 模组重启次数。NVS 中数据不会因为模组重启而丢失，因此只有将这一值存储于 NVS 中，才能起到重启次数计数器的作用。

该示例也演示了如何检测读取/写入操作是否成功，以及某个特定值是否在 NVS 中尚未初始化。诊断程序以纯文本形式提供，帮助您追踪程序流程，及时发现问题。

storage/nvs_rw_blob

演示如何读取及写入 NVS 单个整数值和 BLOB（二进制大对象），并在 NVS 中存储这一数值，即使 ESP32 模组重启也不会消失。

- value - 记录 ESP32 模组软重启次数和硬重启次数。
- blob - 内含记录模组运行次数的表格。此表格将被从 NVS 读取至动态分配的 RAM 上。每次手动软重启后，表格内运行次数即增加一次，新加的运行次数被写入 NVS。下拉 GPIO0 即可手动软重启。

该示例也演示了如何执行诊断程序以检测读取/写入操作是否成功。

storage/nvs_rw_value_cxx

这个例子与 storage/nvs_rw_value 完全一样，只是使用了 C++ 的 NVS 枚举类。

内部实现

键值对日志 NVS 按顺序存储键值对，新的键值对添加在最后。因此，如需更新某一键值对，实际是在日志最后增加一对新的键值对，同时将旧的键值对标记为已擦除。
页面和条目 NVS 库在其操作中主要使用两个实体：页面和条目。页面是一个逻辑结构，用于存储部分的系统日志。逻辑页面对应 flash 的一个物理扇区，正在使用的页面具有与之相关联的序列号。序列号赋予了页面顺序，较高的序列号对应较早创建的页面。页面有以下几种状态：

空或未初始化 页面对应的 flash 扇区为空白状态（所有字节均为 0xff）。此时，页面未存储任何数据且没有关联的序列号。

活跃状态 此时 flash 已完成初始化，页头部写入 flash，页面已具备有效序列号。页面中存在一些空条目，可写入数据。任意时刻，至多有一个页面处于活跃状态。

写满状态 Flash 已写满键值对，状态不再改变。用户无法向写满状态下的页面写入键值对，但仍可将一些键值对标记为已擦除。

擦除状态 未擦除的键值对将移至其他页面，以便擦除当前页面。这一状态仅用于临时状态，即 API 调用返回时，页面应脱离这一状态。如果设备突然断电，下次开机时，设备将继续把未擦除的键值对移至其他页面，并继续擦除当前页面。

损坏状态 页头部包含无效数据，无法进一步解析该页面中的数据，因此之前写入该页面的所有条目均无法访问。相应的 flash 扇区并不会被立即擦除，而是与其他处于未初始化状态的扇区一起等待后续使用。这一状态可能对调试有用。

Flash 扇区映射至逻辑页面并没有特定的顺序，NVS 库会检查存储在 flash 扇区的页面序列号，并根据序列号组织页面。

<table>
<thead>
<tr>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
<th>Page 4</th>
<th>&lt;- 状态</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>Full</td>
<td>Active</td>
<td>Empty</td>
<td>&lt;- 序列号</td>
</tr>
<tr>
<td>#11</td>
<td>#12</td>
<td>#14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+ +-----+ +-----+ +-----+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

页面结构 当前，我们假设 flash 扇区大小为 4096 字节，并且 ESP32 flash 加密命令在 32 字节块上运行。未来有可能引入一些编译时可配置项（可通过 menuconfig 进行配置），以适应具有不同扇区大小的 flash 芯片。但目前尚不清楚 SPI flash 驱动和 SPI flash cache 之类的系统组件是否支持其他扇区大小。

页面由头部、条目状态位图和条目三部分组成。为了实现与 ESP32 flash 加密功能兼容，条目大小设置为 32 字节。如果键值为整数型，条目则保存一个键值对；如果键值为字符串或 BLOB 类型，则条目仅保存一个键值对的部分内容（更多信息请见条目结构描述）。

页面结构如下图所示，括号内数字表示该部分的大小（以字节为单位）。

<table>
<thead>
<tr>
<th>State (4)</th>
<th>Seq. no. (4)</th>
<th>version (1)</th>
<th>Unused (19)</th>
<th>CRC32 (4)</th>
<th>页头部 (32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry state bitmap (32)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entry 0 (32)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entry 1 (32)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entry 125 (32)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

头部和条目状态位图写入 flash 时不加密。如果启用了 ESP32 flash 加密功能，则条目写入 flash 时将会加密。

通过将 0 写入某些位可以定义页面状态值，表示状态改变。因此，如果需要变更页面状态，并不一定要擦除页面，除非要将其更改为 擦除状态。
Chapter 2. API 参考

头部中的 `version` 字段反映了所用的 NVS 格式版本。为实现向后兼容，版本升级从 `0xff` 开始依次递减（例如，`version-1` 为 `0xff`，`version-2` 为 `0xfe`，以此类推）。

头部中 `CRC32` 值是由不包含状态值的条目计算所得（4 到 28 字节）。当前未使用的条目用 `0xff` 字节填充。

条目结构和条目状态位图的详细信息见下文描述。

条目和条目状态位图 每个条目处于以下三种状态之一，每个状态在条目状态位图中用两位表示。位图中的最后两位 (256 - 2 * 126) 未使用。

空（2’ b11）条目还未写入任何内容，处于未初始化状态（全部字节为 `0xff`）。

写入（2’ b10）一个键值对（或跨多个条目的键值对的部分内容）已写入条目中。

擦除（2’ b00）条目中的键值对已丢弃，条目内容不再解析。

条目结构 如果键值类型为基础类型，即 1 - 8 个字节长度的整型数，条目将保存一个键值对；如果键值类型为字符串或 BLOB 类型，条目将保存整个键值对的部分内容。另外，如果键值为字符串类型且跨多个条目，则键值所跨的所有条目均保存在同一页面。BLOB 则可以切分为多个块，实现跨多个页面。

BLOB 索引是一个附加的固定长度元数据条目，用于追踪 BLOB 块。目前条目仍支持早期 BLOB 格式（可读取可修改），但这些 BLOB 一经修改，则以新格式储存至条目。

条目结构中各个字段含义如下：

命令空间（NS, NameSpace）该条目的命名空间索引，详细信息参见命名空间实现章节。

类型（Type）一个字节表示的值的数据类型，`nvs_flash/include/nvs_handle.hpp` 下的 `ItemType` 枚举了可能的类型。

跨度（Span）该键值对所用的条目数量。如果键值为整数型，条目数量即为 1，如果键值为字符串或 BLOB，则条目数量取决于值的长度。

块索引（ChunkIndex）用于存储 BLOB 类型数据块的索引。如果键值为其他数据类型，则此处索引应写入 `0xff`。

CRC32 对条目下所有字节进行校验后，所得的结果和（CRC32 字段不计算在内）。

键（Key）即以结尾的 ASCII 字符串，字符串最长为 15 字节，不包含最后一个字节的零终止符。

数据（Data）如果键值类型为整数型，则数据字段仅包含键值。如果键值小于 8 个字节，使用 `0xff` 填充未使用的部分（右侧）。如果键值类型为 BLOB 索引条目，则该字段的八个字节将保存以下数据块信息：

- 块大小 整个 BLOB 数据的大小（以字节为单位）。该字段仅用于 BLOB 索引类型条目。
- **ChunkCount** 存储过程中 BLOB 分成的数据块总量，该字段仅用于 BLOB 索引类型条目。
- **ChunkStart** BLOB 第一个数据块的块开始，后续数据块索引依次递增，步长为 1。该字段仅用于 BLOB 索引类型条目。

如果键值类型为字符串或 BLOB 数据块，数据字段的这八个字节将保存该键值的一些附加信息，如下所示：
- **数据大小** 实际数据的大小（以字节为单位）。如果键值类型为字符串，此字段也应将终止符包含在内。此字段仅用于字符串和 BLOB 类型条目。
- **CRC32** 数据所有字段的校验和，该字段仅用于字符串和 BLOB 类型条目。

可变长度值（字符串和 BLOB）写入后续条目，每个条目 32 字节。第一个条目的 **Span** 字段将指明使用了多少条目。

**命名空间** 如上所述，每个键值对属于一个命名空间。命名空间标识符（字符串）也作为键值对的键，存储在索引为 0 的命名空间中。与这些键对应的值就是这些命名空间的索引。

| NS=0 Type=uint8_t Key="wifi" Value=1 | Entry describing namespace "wifi" |
| NS=1 Type=uint32_t Key="channel" Value=6 | Key "channel" in namespace "wifi" |
| NS=0 Type=uint8_t Key="pwm" Value=2 | Entry describing namespace "pwm" |
| NS=2 Type=uint16_t Key="channel" Value=20 | Key "channel" in namespace "pwm" |

**条目哈希列表** 为了减少对 flash 执行的读操作次数，Page 类对象均设有一个列表，包含一对数据：条目索引和条目哈希值。该列表可大大提高检索速度，而无需迭代所有条目并逐个从 flash 中读取。

Page::findItem 首先从哈希列表中检索条目哈希值。如果条目存在，则在页面内给出条目索引。由于哈希冲突，在哈希列表中检索条目哈希值可能会得到不同的条目，对 flash 中条目再次迭代可解决这一冲突。

哈希列表中每个节点均包含一个 24 位哈希值和 8 位条目索引。哈希值根据条目命名空间、键名和块索引由 CRC32 计算所得，计算结果保留 24 位。为减少将 32 位条目存储在链表中的开销，链表采用了数组的双向链表。每个数组占用 128 个字节，包含 29 个条目，两个链表指针和一个 32 位计数组段。因此，每页额外需要的 RAM 最少为 128 字节，最多为 640 字节。

**API 参考**

**Header File**
- components/nvs_flash/include/nvs_flash.h

**Functions**

```c
esp_err_t nvs_flash_init(void)
```

Initializes the default NVS partition.

This API initialises the default NVS partition. The default NVS partition is the one that is labeled “nvs” in the partition table.

When “NVS_ENCRYPTION” is enabled in the menuconfig, this API enables the NVS encryption for the default NVS partition as follows

a. Read security configurations from the first NVS key partition listed in the partition table. (NVS key partition is any “data” type partition which has the subtype value set to “nvs_keys”)

b. If the NVS key partition obtained in the previous step is empty, generate and store new keys in that NVS key partition.

c. Internally call “nvs_flash_secure_init()” with the security configurations obtained/generated in the previous steps.
Post initialization NVS read/write APIs remain the same irrespective of NVS encryption.

初始化
- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if no partition with label “nvs” is found in the partition table
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver
- error codes from nvs_flash_read_security_cfg API (when “NVS_ENCRYPTION” is enabled).
- error codes from nvs_flash_generate_keys API (when “NVS_ENCRYPTION” is enabled).
- error codes from nvs_flash_secure_init_partition API (when “NVS_ENCRYPTION” is enabled).

```
esp_err_t nvs_flash_init_partition (const char* partition_label)

Initialize NVS flash storage for the specified partition.
```

参数 `partition_label` – [in] Label of the partition. Must be no longer than 16 characters.

返回
- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if specified partition is not found in the partition table
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver

```
esp_err_t nvs_flash_init_partition_ptr (const esp_partition_t* partition)

Initialize NVS flash storage for the partition specified by partition pointer.
```

参数 `partition` – [in] pointer to a partition obtained by the ESP partition API.

返回
- ESP_OK if storage was successfully initialized
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_INVALID_ARG in case partition is NULL
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver

```
esp_err_t nvs_flash_deinit (void)

Deinitialize NVS storage for the default NVS partition.
```

Default NVS partition is the partition with “nvs” label in the partition table.

返回
- ESP_OK on success (storage was deinitialized)
- ESP_ERR_NVS_NOT_INITIALIZED if the storage was not initialized prior to this call

```
esp_err_t nvs_flash_deinit_partition (const char* partition_label)

Deinitialize NVS storage for the given NVS partition.
```

参数 `partition_label` – [in] Label of the partition

返回
- ESP_OK on success
- ESP_ERR_NVS_NOT_INITIALIZED if the storage for given partition was not initialized prior to this call

```
esp_err_t nvs_flash_erase (void)

Erase the default NVS partition.
```

Erases all contents of the default NVS partition (one with label “nvs”).
Chapter 2. API

If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.

**esp_err_t nvs_flash_erase_partition (const char *part_name)**

Erase specified NVS partition.

Erase all content of a specified NVS partition.

**esp_err_t nvs_flash_erase_partition_ptr (const esp_partition_t *partition)**

Erase custom partition.

Erase all content of specified custom partition.

**esp_err_t nvs_flash_secure_init (nvs_sec_cfg_t *cfg)**

Initialize the default NVS partition.

This API initialises the default NVS partition. The default NVS partition is the one that is labeled “nvs” in the partition table.

**参数** part_name – [in] Name (label) of the partition which should be erased

**返回**
- ESP_OK on success
- ESP_ERR_NOT_FOUND if there is no NVS partition labeled “nvs” in the partition table
- different error in case de-initialization fails (shouldn’t happen)

**参数** partition – [in] pointer to a partition obtained by the ESP partition API.

**返回**
- ESP_OK on success
- ESP_ERR_NOT_FOUND if there is no partition with the specified parameters in the partition table
- ESP_ERR_INVALID_ARG in case partition is NULL
- one of the error codes from the underlying flash storage driver

**备注:** If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.
Chapter 2. API

one of the error codes from the underlying flash storage driver

`esp_err_t nvs_flash_secure_init_partition` (const char *partition_label, `nvs_sec_cfg_t` *cfg)

Initialize NVS flash storage for the specified partition.

**参数**

- **partition_label** – [in] Label of the partition. Note that internally, a reference to passed value is kept and it should be accessible for future operations
- **cfg** – [in] Security configuration (keys) to be used for NVS encryption/decryption. If cfg is null, no encryption/decryption is used.

**返回**

- ESP_OK if storage has been initialized successfully.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if specified partition is not found in the partition table
- ESP_ERR_NO_MEM in case memory could not be allocated for the internal structures
- one of the error codes from the underlying flash storage driver

`esp_err_t nvs_flash_generate_keys` (const `esp_partition_t` *partition, `nvs_sec_cfg_t` *cfg)

Generate and store NVS keys in the provided esp partition.

**参数**

- **partition** – [in] Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- **cfg** – [out] Pointer to nvs security configuration structure. Pointer must be non-NULL. Generated keys will be populated in this structure.

**返回**

- ESP_OK, if cfg was read successfully; -ESP_INVALID_ARG, if partition or cfg; - or error codes from esp_partition_write/erase APIs.

`esp_err_t nvs_flash_read_security_cfg` (const `esp_partition_t` *partition, `nvs_sec_cfg_t` *cfg)

Read NVS security configuration from a partition.

备注：Provided partition is assumed to be marked ‘encrypted’.

**Structures**

`struct nvs_sec_cfg_t`

Key for encryption and decryption.

**Public Members**

- `uint8_t eky[NVS_KEY_SIZE]`
  XTS encryption and decryption key
- `uint8_t tky[NVS_KEY_SIZE]`
  XTS tweak key
Chapter 2. API

Macros

**NVS_KEY_SIZE**

Header File

- components/nvs_flash/include/nvs.h

Functions

**esp_err_t nvs_set_i8** (*nvs_handle_t handle, const char *key, int8_t value*)

set int8_t value for given key

Set value for the key, given its name. Note that the actual storage will not be updated until **nvs_commit** is called.

- **handle** - [in] Handle obtained from **nvs_open** function. Handles that were opened read only cannot be used.
- **key** - [in] Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’t be empty.
- **value** - [in] The value to set.

- ESP_OK if value was set successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
- ESP_ERR_NVS_INVALID_NAME if key name doesn’t satisfy constraints
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
- ESP_ERR_NVS_REMOVE_FAILED if the value wasn’t updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of **nvs**, provided that flash operation doesn’t fail again.

**esp_err_t nvs_set_u8** (*nvs_handle_t handle, const char *key, uint8_t value*)

set uint8_t value for given key

This function is the same as **nvs_set_i8** except for the data type.

**esp_err_t nvs_set_i16** (*nvs_handle_t handle, const char *key, int16_t value*)

set int16_t value for given key

This function is the same as **nvs_set_i8** except for the data type.

**esp_err_t nvs_set_u16** (*nvs_handle_t handle, const char *key, uint16_t value*)

set uint16_t value for given key

This function is the same as **nvs_set_i8** except for the data type.

**esp_err_t nvs_set_i32** (*nvs_handle_t handle, const char *key, int32_t value*)

set int32_t value for given key

This function is the same as **nvs_set_i8** except for the data type.

**esp_err_t nvs_set_u32** (*nvs_handle_t handle, const char *key, uint32_t value*)

set uint32_t value for given key

This function is the same as **nvs_set_i8** except for the data type.

**esp_err_t nvs_set_i64** (*nvs_handle_t handle, const char *key, int64_t value*)

set int64_t value for given key

This function is the same as **nvs_set_i8** except for the data type.
**Chapter 2. API**

*esp_err_t* `nvs_set_u64` *(nvs_handle_t handle, const char* key, uint64_t value)*

Set uint64_t value for given key

This function is the same as `nvs_set_i8` except for the data type.

*esp_err_t* `nvs_set_str` *(nvs_handle_t handle, const char* key, const char* value)*

Set string for given key

Set value for the key, given its name. Note that the actual storage will not be updated until `nvs_commit` is called.

**参数**

- **handle** - [in] Handle obtained from `nvs_open` function. Handles that were opened read only cannot be used.
- **key** - [in] Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’t be empty.
- **value** - [in] The value to set. For strings, the maximum length (including null character) is 4000 bytes, if there is one complete page free for writing. This decreases, however, if the free space is fragmented.

**返回**

- ESP_OK if value was set successfully
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
- ESP_ERR_NVS_INVALID_NAME if key name doesn’t satisfy constraints
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
- ESP_ERR_NVS_REMOVE_FAILED if the value wasn’t updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn’t fail again.
- ESP_ERR_NVS_VALUE_TOO_LONG if the string value is too long

*esp_err_t* `nvs_get_i8` *(nvs_handle_t handle, const char* key, int8_t*out_value)*

Get int8_t value for given key

These functions retrieve value for the key, given its name. If `key` does not exist, or the requested variable type doesn’t match the type which was used when setting a value, an error is returned.

In case of any error, `out_value` is not modified.

`out_value` has to be a pointer to an already allocated variable of the given type.

```c
// Example of using nvs_get_i32:
int32_t max_buffer_size = 4096; // default value
esp_err_t err = nvs_get_i32(my_handle, "max_buffer_size", &max_buffer_size);
assert(err == ESP_OK || err == ESP_ERR_NVS_NOT_FOUND);
// if ESP_ERR_NVS_NOT_FOUND was returned, max_buffer_size will still have its default value.
```

**参数**

- **handle** - [in] Handle obtained from `nvs_open` function.
- **key** - [in] Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’t be empty.
- **out_value** - Pointer to the output value. May be NULL for `nvs_get_str` and `nvs_get_blob`, in this case required length will be returned in length argument.

**返回**

- ESP_OK if the value was retrieved successfully
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn’t exist
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL

Espressif Systems 1661 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
• ESP_ERR_NVS_INVALID_NAME if key name doesn’t satisfy constraints
• ESP_ERR_NVS_INVALID_LENGTH if length is not sufficient to store data

```c
esp_err_t nvs_get_u8 (nvs_handle_t handle, const char *key, uint8_t *out_value)
  get uint8_t value for given key
  This function is the same as nvs_get_i8 except for the data type.

esp_err_t nvs_get_i16 (nvs_handle_t handle, const char *key, int16_t *out_value)
  get int16_t value for given key
  This function is the same as nvs_get_i8 except for the data type.

esp_err_t nvs_get_u16 (nvs_handle_t handle, const char *key, uint16_t *out_value)
  get uint16_t value for given key
  This function is the same as nvs_get_i8 except for the data type.

esp_err_t nvs_get_i32 (nvs_handle_t handle, const char *key, int32_t *out_value)
  get int32_t value for given key
  This function is the same as nvs_get_i8 except for the data type.

esp_err_t nvs_get_u32 (nvs_handle_t handle, const char *key, uint32_t *out_value)
  get uint32_t value for given key
  This function is the same as nvs_get_i8 except for the data type.

esp_err_t nvs_get_i64 (nvs_handle_t handle, const char *key, int64_t *out_value)
  get int64_t value for given key
  This function is the same as nvs_get_i8 except for the data type.

esp_err_t nvs_get_u64 (nvs_handle_t handle, const char *key, uint64_t *out_value)
  get uint64_t value for given key
  This function is the same as nvs_get_i8 except for the data type.

esp_err_t nvs_get_str (nvs_handle_t handle, const char *key, char *out_value, size_t *length)
  get string value for given key
  These functions retrieve the data of an entry, given its key. If key does not exist, or the requested variable type
doesn’t match the type which was used when setting a value, an error is returned.
  In case of any error, out_value is not modified.

All functions expect out_value to be a pointer to an already allocated variable of the given type.

nvs_get_str and nvs_get_blob functions support WinAPI-style length queries. To get the size necessary to
store the value, call nvs_get_str or nvs_get_blob with zero out_value and non-zero pointer to length. Variable
pointed to by length argument will be set to the required length. For nvs_get_str, this length includes the zero
terminator. When calling nvs_get_str and nvs_get_blob with non-zero out_value, length has to be non-zero and
has to point to the length available in out_value. It is suggested that nvs_get/set_str is used for zero-terminated
C strings, and nvs_get/set_blob used for arbitrary data structures.
```

```c
// Example (without error checking) of using nvs_get_str to get a string into
  a dynamic array:
  size_t required_size;
  nvs_get_str(my_handle, "server_name", NULL, &required_size);
  char* server_name = malloc(required_size);
  nvs_get_str(my_handle, "server_name", server_name, &required_size);

// Example (without error checking) of using nvs_get_blob to get a binary data
  into a static array:
```

(下页继续)
uint8_t mac_addr[6];
size_t size = sizeof(mac_addr);
nvs_get_blob(my_handle, "dst_mac_addr", mac_addr, &size);

参数
- **handle** - [in] Handle obtained from nvs_open function.
- **key** - [in] Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’t be empty.
- **out_value** - [out] Pointer to the output value. May be NULL for nvs_get_str and nvs_get_blob, in this case required length will be returned in length argument.
- **length** - [inout] A non-zero pointer to the variable holding the length of out_value. In case out_value a zero, will be set to the length required to hold the value. In case out_value is not zero, will be set to the actual length of the value written. For nvs_get_str this includes zero terminator.

返回
- **ESP_OK** if the value was retrieved successfully
- **ESP_FAIL** if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- **ESP_ERR_NVS_NOT_FOUND** if the requested key doesn’t exist
- **ESP_ERR_NVS_INVALID_HANDLE** if handle has been closed or is NULL
- **ESP_ERR_NVS_INVALID_NAME** if key name doesn’t satisfy constraints
- **ESP_ERR_NVS_INVALID_LENGTH** if length is not sufficient to store data

`esp_err_t nvs_get_blob(nvs_handle_t handle, const char*key, void*out_value, size_t*length)`
get blob value for given key

This function behaves the same as nvs_get_str, except for the data type.

`esp_err_t nvs_open(const char*namespace_name, nvs_open_mode_t open_mode, nvs_handle_t*out_handle)`
Open non-volatile storage with a given namespace from the default NVS partition.

Multiple internal ESP-IDF and third party application modules can store their key-value pairs in the NVS module. In order to reduce possible conflicts on key names, each module can use its own namespace. The default NVS partition is the one that is labelled “nvs” in the partition table.

参数
- **namespace_name** - [in] Namespace name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’t be empty.
- **open_mode** - [in] NVS_READWRITE or NVS_READONLY. If NVS_READONLY, will open a handle for reading only. All write requests will be rejected for this handle.
- **out_handle** - [out] If successful (return code is zero), handle will be returned in this argument.

返回
- **ESP_OK** if storage handle was opened successfully
- **ESP_FAIL** if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- **ESP_ERR_NVS_NOT_INITIALIZED** if the storage driver is not initialized
- **ESP_ERR_NVS_PART_NOT_FOUND** if the partition with label “nvs” is not found
- **ESP_ERR_NVS_NOT_FOUND** if namespace doesn’t exist yet and mode is NVS_READONLY
- **ESP_ERR_NVS_INVALID_NAME** if namespace name doesn’t satisfy constraints
- **ESP_ERR_NO_MEM** in case memory could not be allocated for the internal structures
- **ESP_ERR_NVS_NOT_ENOUGH_SPACE** if there is no space for a new entry or there are too many different namespaces (maximum allowed different namespaces: 254)
- other error codes from the underlying storage driver

`esp_err_t nvs_open_from_partition(const char*part_name, const char*namespace_name, nvs_open_mode_t open_mode, nvs_handle_t*out_handle)`
Open non-volatile storage with a given namespace from specified partition.

The behaviour is same as nvs_open() API. However this API can operate on a specified NVS partition instead of default NVS partition. Note that the specified partition must be registered with NVS using nvs_flash_init_partition() API.

参数
• part_name – [in] Label (name) of the partition of interest for object read/write/erase
• namespace_name – [in] Namespace name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’ t be empty.
• open_mode – [in] NVS_READWRITE or NVS_READONLY. If NVS_READONLY, will open a handle for reading only. All write requests will be rejected for this handle.
• out_handle – [out] If successful (return code is zero), handle will be returned in this argument.

返回
• ESP_OK if storage handle was opened successfully
• ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
• ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized
• ESP_ERR_NVS_PART_NOT_FOUND if the partition with specified name is not found
• ESP_ERR_NVS_NOT_FOUND id namespace doesn’ t exist yet and mode is NVS_READONLY
• ESP_ERR_NVS_INVALID_NAME if namespace name doesn’t satisfy constraints
• ESP_ERR_NVS_READ_ONLY if storage handle was opened as readonly
• ESP_ERR_NVS_VALUE_TOO_LONG if the value is too long

esp_err_t nvs_set_blob (nvs_handle_t handle, const char* key, const void* value, size_t length)

set variable length binary value for given key

This family of functions set value for the key, given its name. Note that actual storage will not be updated until nvs_commit function is called.

参数
• handle – [in] Handle obtained from nvs_open function. Handles that were opened read only cannot be used.
• key – [in] Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’ t be empty.
• value – [in] The value to set.
• length – [in] length of binary value to set, in bytes; Maximum length is 508000 bytes or (97.6% of the partition size - 4000) bytes whichever is lower.

返回
• ESP_OK if value was set successfully
• ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
• ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
• ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
• ESP_ERR_NVS_INVALID_NAME if key name doesn’ t satisfy constraints
• ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
• ESP_ERR_NVS_REMOVE_FAILED if the value wasn’ t updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn’ t fail again.
• ESP_ERR_NVS_VALUE_TOO_LONG if the value is too long

esp_err_t nvs_erase_key (nvs_handle_t handle, const char *key)

Erase key-value pair with given key name.

Note that actual storage may not be updated until nvs_commit function is called.
### Chapter 2. API 参考

#### 参数
- **handle** - [in] Storage handle obtained with `nvs_open`. Handles that were opened read only cannot be used.
- **key** - [in] Key name. Maximum length is (NVS_KEY_NAME_MAX_SIZE-1) characters. Shouldn’t be empty.

#### 返回
- ESP_OK if erase operation was successful
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if handle was opened as read only
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn’t exist
- other error codes from the underlying storage driver

```c
esp_err_t nvs_erase_all(nvs_handle_t handle)
```

Erase all key-value pairs in a namespace.

Note that actual storage may not be updated until `nvs_commit` function is called.

- **handle** - [in] Storage handle obtained with `nvs_open`. Handles that were opened read only cannot be used.

#### 返回
- ESP_OK if erase operation was successful
- ESP_FAIL if there is an internal error; most likely due to corrupted NVS partition (only if NVS assertion checks are disabled)
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if handle was opened as read only
- other error codes from the underlying storage driver

```c
esp_err_t nvs_commit(nvs_handle_t handle)
```

Write any pending changes to non-volatile storage.

After setting any values, `nvs_commit()` must be called to ensure changes are written to non-volatile storage. Individual implementations may write to storage at other times, but this is not guaranteed.

- **handle** - [in] Storage handle obtained with `nvs_open`. Handles that were opened read only cannot be used.

#### 返回
- ESP_OK if the changes have been written successfully
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- other error codes from the underlying storage driver

```c
void nvs_close(nvs_handle_t handle)
```

Close the storage handle and free any allocated resources.

This function should be called for each handle opened with `nvs_open` once the handle is not in use any more. Closing the handle may not automatically write the changes to nonvolatile storage. This has to be done explicitly using `nvs_commit` function. Once this function is called on a handle, the handle should no longer be used.

- **handle** - [in] Storage handle to close

```c
esp_err_t nvs_get_stats(const char *part_name, nvs_stats_t *nvs_stats)
```

Fill structure `nvs_stats_t`. It provides info about used memory the partition.

This function calculates to runtime the number of used entries, free entries, total entries, and amount namespace in partition.

```c
// Example of nvs_get_stats() to get the number of used entries and free_entries:

nvs_stats_t nvs_stats;

nvs_get_stats(NULL, &nvs_stats);
```

(下页继续)
printf("Count: UsedEntries = (%d), FreeEntries = (%d), AllEntries = (%d)\n", 
    nvs_stats.used_entries, nvs_stats.free_entries, nvs_stats.total_entries);

参数

- **part_name**  - [in] 党同名 NVS 在分区表中。如果不传递 NULL，则将使用 NVS_DEFAULT_PART_NAME（“nvs”）。
- **nvs_stats**  - [out] 返回填充结构 nvs_states_t。它提供有关使用内存的分区信息。

返回

- ESP_OK 如果更改写入成功。返回参数 nvs_stats 将被填充。
- ESP_ERR_NVS_PART_NOT_FOUND 如果带有标签 “name” 的分区未找到。返回参数 nvs_stats 将被填充 0。
- ESP_ERR_NVS_NOT_INITIALIZED 如果存储器驱动未初始化。返回参数 nvs_stats 将被填充 0。
- ESP_ERR_INVALID_ARG 如果 nvs_stats 为 NULL。
- ESP_ERR_INVALID_STATE 如果存在无效状态的页面。由于 not all pages will be counted. Counting will be interrupted at the first INVALID page。

```c
esp_err_t nvs_get_used_entry_count (nvs_handle_t handle, size_t *used_entries)
```

计数所有命名空间的条目。

一个条目表示 NVS 中的最小存储单元。字符串和 blob 可能占用多个条目。

注意，要找出命名空间中已占用的条目总数，请将返回值 used_entries（如果 err 等于 ESP_OK）。因为命名空间条目只占用一个条目。

```c
// Example of nvs_get_used_entry_count() to get amount of all key-value pairs... 
in one namespace:
    nvs_handle_t handle;
    nvs_open("namespace1", NVS_READWRITE, &handle);
    ...
    size_t used_entries;
    size_t total_entries_namespace;
    if (nvs_get_used_entry_count(handle, &used_entries) == ESP_OK){
        // the total number of entries occupied by the namespace
        total_entries_namespace = used_entries + 1;
    }
```

参数

- **handle**  - [in] 从 nvs_open 函数获得的句柄。
- **used_entries**  - [out] 返回命名空间中已占用的条目数。

返回

- ESP_OK 如果更改已写入成功。返回参数 used_entries 将被填充有效值。
- ESP_ERR_NVS_NOT_INITIALIZED 如果存储器驱动未初始化。返回参数 used_entries 将被填充 0。
- ESP_ERR_NVS_INVALID_HANDLE 如果 handle 已关闭或为 NULL。返回参数 used_entries 将被填充 0。
- ESP_ERR_INVALID_ARG 如果 used_entries 为 NULL。
- 其他错误代码来自底层存储驱动。返回参数 used_entries 将被填充 0。

```c
esp_err_t nvs_entry_find (const char *part_name, const char *namespace_name, nvs_type_t type, 
    nvs_iterator_t *output_iterator)
```
Create an iterator to enumerate NVS entries based on one or more parameters.

```c
// Example of listing all the key-value pairs of any type under specified
// partition and namespace
nvs_iterator_t it = NULL;
esp_err_t res = nvs_entry_find(<nvs_partition_name>, <namespace>, NVS_TYPE_--ANY, &it);
while(res == ESP_OK) {
    nvs_entry_info_t info;
    nvs_entry_info(it, &info); // Can omit error check if parameters are
    // guaranteed to be non-NULL
    printf("key '%s', type '%d' \n", info.key, info.type);
    res = nvs_entry_next(&it);
}
nvs_release_iterator(it);
```

**参数**
- **part_name** - [in] Partition name
- **namespace_name** - [in] Set this value if looking for entries with a specific namespace. Pass NULL otherwise.
- **type** - [in] One of nvs_type_t values.
- **output_iterator** - [out] Set to a valid iterator to enumerate all the entries found. Set to NULL if no entry for specified criteria was found. If any other error except ESP_ERR_INVALID_ARG occurs, output_iterator is NULL, too. If ESP_ERR_INVALID_ARG occurs, output_iterator is not changed. If a valid iterator is obtained through this function, it has to be released using nvs_release_iterator when not used any more, unless ESP_ERR_INVALID_ARG is returned.

**返回**
- ESP_OK if no internal error or programming error occurred.
- ESP_ERR_NVST_NOT_FOUND if no element of specified criteria has been found.
- ESP_ERR_NO_MEM if memory has been exhausted during allocation of internal structures.
- ESP_ERR_INVALID_ARG if any of the parameters is NULL. Note: don’t release output_iterator in case ESP_ERR_INVALID_ARG has been returned.

**esp_err_t nvs_entry_next (nvs_iterator_t *iterator)**

Advances the iterator to next item matching the iterator criteria.

Note that any copies of the iterator will be invalid after this call.

**参数**
- **iterator** - [inout] Iterator obtained from nvs_entry_find function. Must be non-NULL. If any error except ESP_ERR_INVALID_ARG occurs, iterator is set to NULL. If ESP_ERR_INVALID_ARG occurs, iterator is not changed.

**返回**
- ESP_OK if no internal error or programming error occurred.
- ESP_ERR_NVST_NOT_FOUND if no next element matching the iterator criteria.
- ESP_ERR_INVALID_ARG if iterator is NULL.
- Possibly other errors in the future for internal programming or flash errors.

**esp_err_t nvs_entry_info (const nvs_iterator_t iterator, nvs_entry_info_t *out_info)**

Fills nvs_entry_info_t structure with information about entry pointed to by the iterator.

**参数**
- **iterator** - [in] Iterator obtained from nvs_entry_find function. Must be non-NULL.
- **out_info** - [out] Structure to which entry information is copied.

**返回**
- ESP_OK if all parameters are valid; current iterator data has been written to out_info.
```c
void nvs_release_iterator(nvs_iterator_t iterator)

参数 iterator - [in] Release iterator obtained from nvs_entry_find function. NULL argument is allowed.
```

### Structures

```c
struct nvs_entry_info_t
```

- Information about entry obtained from nvs_entry_info function

#### Public Members

- char namespace_name[NVS_NS_NAME_MAX_SIZE]
  - Namespace to which key-value belong

- char key[NVS_KEY_NAME_MAX_SIZE]
  - Key of stored key-value pair

- nvs_type_t type
  - Type of stored key-value pair

- struct nvs_stats_t

  备注:  Info about storage space NVS.

#### Public Members

- size_t used_entries
  - Amount of used entries.

- size_t free_entries
  - Amount of free entries.

- size_t total_entries
  - Amount all available entries.

- size_t namespace_count
  - Amount name space.

### Macros

- **ESP_ERR_NVS_BASE**
  - Starting number of error codes

- **ESP_ERR_NVS_NOT_INITIALIZED**
  - The storage driver is not initialized
ESP_ERR_NVS_NOT_FOUND
A requested entry couldn’t be found or namespace doesn’t exist yet and mode is NVS_READONLY

ESP_ERR_NVS_TYPE_MISMATCH
The type of set or get operation doesn’t match the type of value stored in NVS

ESP_ERR_NVS_READ_ONLY
Storage handle was opened as read only

ESP_ERR_NVS_NOT_ENOUGH_SPACE
There is not enough space in the underlying storage to save the value

ESP_ERR_NVS_INVALID_NAME
Namespace name doesn’t satisfy constraints

ESP_ERR_NVS_INVALID_HANDLE
Handle has been closed or is NULL

ESP_ERR_NVS_REMOVE_FAILED
The value wasn’t updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn’t fail again.

ESP_ERR_NVS_KEY_TOO_LONG
Key name is too long

ESP_ERR_NVS_PAGE_FULL
Internal error; never returned by nvs API functions

ESP_ERR_NVS_INVALID_STATE
NVS is in an inconsistent state due to a previous error. Call nvs_flash_init and nvs_open again, then retry.

ESP_ERR_NVS_INVALID_LENGTH
String or blob length is not sufficient to store data

ESP_ERR_NVS_NO_FREE_PAGES
NVS partition doesn’t contain any empty pages. This may happen if NVS partition was truncated. Erase the whole partition and call nvs_flash_init again.

ESP_ERR_NVS_VALUE_TOO_LONG
Value doesn’t fit into the entry or string or blob length is longer than supported by the implementation

ESP_ERR_NVS_PART_NOT_FOUND
Partition with specified name is not found in the partition table

ESP_ERR_NVS_NEW_VERSION_FOUND
NVS partition contains data in new format and cannot be recognized by this version of code

ESP_ERR_NVS_XTS_ENCR_FAILED
XTS encryption failed while writing NVS entry
Chapter 2. API

ESP_ERR_NVS_XTS_DECR_FAILED
XTS decryption failed while reading NVS entry

ESP_ERR_NVS_XTS_CFG_FAILED
XTS configuration setting failed

ESP_ERR_NVS_XTS_CFG_NOT_FOUND
XTS configuration not found

ESP_ERR_NVS_ENCR_NOT_SUPPORTED
NVS encryption is not supported in this version

ESP_ERR_NVS_KEYS_NOT_INITIALIZED
NVS key partition is uninitialized

ESP_ERR_NVS_CORRUPT_KEY_PART
NVS key partition is corrupt

ESP_ERR_NVS_WRONG_ENCRYPTION
NVS partition is marked as encrypted with generic flash encryption. This is forbidden since the NVS encryption works differently.

ESP_ERR_NVS_CONTENT_DIFFERS
Internal error; never returned by nvs API functions. NVS key is different in comparison

NVS_DEFAULT_PART_NAME
Default partition name of the NVS partition in the partition table

NVS_PART_NAME_MAX_SIZE
maximum length of partition name (excluding null terminator)

NVS_KEY_NAME_MAX_SIZE
Maximum length of NVS key name (including null terminator)

NVS_NS_NAME_MAX_SIZE
Maximum length of NVS namespace name (including null terminator)

Type Definitions

typedef uint32_t nvs_handle_t
Opaque pointer type representing non-volatile storage handle

typedef nvs_handle_t nvs_handle

typedef nvs_open_mode_t nvs_open_mode

typedef struct nvs_opaque_iterator_t *nvs_iterator_t
Opaque pointer type representing iterator to nvs entries
Enumerations

enum nvs_open_mode_t
    Mode of opening the non-volatile storage.
    
    enumerator NVS_READONLY
        Read only
    
    enumerator NVS_READWRITE
        Read and write

enum nvs_type_t
    Types of variables.
    
    enumerator NVS_TYPE_U8
        Type uint8_t
    
    enumerator NVS_TYPE_I8
        Type int8_t
    
    enumerator NVS_TYPE_U16
        Type uint16_t
    
    enumerator NVS_TYPE_I16
        Type int16_t
    
    enumerator NVS_TYPE_U32
        Type uint32_t
    
    enumerator NVS_TYPE_I32
        Type int32_t
    
    enumerator NVS_TYPE_U64
        Type uint64_t
    
    enumerator NVS_TYPE_I64
        Type int64_t
    
    enumerator NVS_TYPE_STR
        Type string
    
    enumerator NVS_TYPE_BLOB
        Type blob
    
    enumerator NVS_TYPE_ANY
        Must be last
2.9.4 NVS 分区生成程序

介绍

NVS 分区生成程序 (nvs_flash/nvs_partition_generator/nvs_partition_gen.py) 根据 CSV 文件中的键值对生成二进制文件。该二进制文件与非易失性存储器 (NVS) 中定义的 NVS 结构兼容。NVS 分区生成程序适合用于生成二进制数据 (Blob)，其中包括设备生产时可从外部烧录的 ODM/OEM 数据。这也使得生产制造商在使用同一个应用固件的基础上，通过自定义参数，如序列号，为每个设备生成不同配置的二进制 NVS 分区。

准备工作

在加密模式下使用该程序，需安装下列软件包：

- cryptography package

根据目录下的 requirements.txt 包含必需 python 包，请预先安装。

CSV 文件格式

CSV 文件每行需包含四个参数，以逗号隔开。具体参数描述见下表：

<table>
<thead>
<tr>
<th>序号</th>
<th>参数</th>
<th>描述</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Key</td>
<td>主键，应用程序可通过查询此键来获取数据。</td>
<td>file、data 和 namespace。</td>
</tr>
<tr>
<td>2</td>
<td>Type</td>
<td>支持 8、16、32,string,hex2bin、base64 和 binary。决定二进制 bin 文件中 value 被编码成的类型。string 和 binary 编码的区别在于，string 数据以 NULL 字符结尾，binary 数据则不是。</td>
<td>file 类型当前仅支持 hex2bin、base64、string 和 binary 编码。</td>
</tr>
<tr>
<td>3</td>
<td>Encoding</td>
<td>支持 u8、u16、u32、i32、u64、i64、string、hex2bin、base64 和 binary。决定二进制 bin 文件中 value 被编码成的类型。string 和 binary 编码的区别在于，string 数据以 NULL 字符结尾，binary 数据则不是。</td>
<td>namespace 字段的 encoding 和 value 应为空。namespace 的 encoding 和 value 为固定值，不可设置。这些单元格中的所有值都会被忽略。</td>
</tr>
<tr>
<td>4</td>
<td>Value</td>
<td>Data value</td>
<td></td>
</tr>
</tbody>
</table>

备注：CSV 文件的第一行应始终为列标题，不可设置。

此类 CSV 文件的 Dump 示例如下：

```
key, type, encoding, value  <- 列标题
namespace_name, namespace,,  <- 第一个条目为 "namespace"
key1, data, u8, i
key2, file, string, /path/to/file
```

备注：

请确保：

- 逗号 ‘,’ 前后无空格；
- CSV 文件每行末尾无空格。
NVS 条目和命名空间 (namespace) 的关联

如 CSV 文件中出现命名空间条目，后续条目均会被视为该命名空间的一部分，直至找到下一个命名空间条目。找到新命名空间条目后，后续所有条目都为新命名空间的一部分。

备注：CSV 文件中第一个条目应始终为 namespace。

支持多页 Blob

默认情况下，二进制 Blob 可跨多页，格式参考条目结构章节。如需使用旧版格式，可在程序中禁用该功能。

支持加密

NVS 分区生成程序还可使用 AES-XTS 加密生成二进制加密文件。更多信息详见 NVS 加密。

支持解密

如果 NVS 二进制文件采用了 AES-XTS 加密，该程序还可对此类文件进行解密。更多信息详见 NVS 加密。

运行程序

使用方法：

```
python nvs_partition_gen.py [-h] {generate,generate-key,encrypt,decrypt} ...
```

可选参数：

<table>
<thead>
<tr>
<th>序号</th>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-h, --help</td>
<td>显示帮助信息并退出</td>
</tr>
</tbody>
</table>

命令：

运行 nvs_partition_gen.py (command) -h 查看更多帮助信息

<table>
<thead>
<tr>
<th>序号</th>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>generate</td>
<td>生成 NVS 分区</td>
</tr>
<tr>
<td>2</td>
<td>generate-key</td>
<td>生成加密密钥</td>
</tr>
<tr>
<td>3</td>
<td>encrypt</td>
<td>加密 NVS 分区</td>
</tr>
<tr>
<td>4</td>
<td>decrypt</td>
<td>解密 NVS 分区</td>
</tr>
</tbody>
</table>

生成 NVS 分区（默认模式） 使用方法：

```
python nvs_partition_gen.py generate [-h] [--version {1,2}] [--outdir OUTDIR]
```

input output size
Chapter 2. API 参考

位置参数:

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>待解析的 CSV 文件路径</td>
</tr>
<tr>
<td>output</td>
<td>NVS 二进制文件的输出路径</td>
</tr>
<tr>
<td>size</td>
<td>NVS 分区大小（以字节为单位，且为 4096 的整数倍）</td>
</tr>
</tbody>
</table>

可选参数:

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>显示帮助信息并退出</td>
</tr>
<tr>
<td>version</td>
<td>版本为 1.2。</td>
</tr>
<tr>
<td>-outdir OUTDIR</td>
<td>输出目录，用于存储创建的文件。（默认当前目录）</td>
</tr>
</tbody>
</table>

运行如下命令创建 NVS 分区，该程序同时会提供 CSV 示例文件:

```bash
python nvs_partition_gen.py generate_sample_singlepage_blob.csv sample.bin 0x3000
```

仅生成加密钥分区 使用方法:

```bash
python nvs_partition_gen.py generate-key [-h] [--keyfile KEYFILE] [--outdir OUTDIR]
```

可选参数:

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>显示帮助信息并退出</td>
</tr>
<tr>
<td>-keyfile KEYFILE</td>
<td>加密密钥分区文件的输出路径</td>
</tr>
<tr>
<td>-outdir OUTDIR</td>
<td>输出目录，用于存储创建的文件（默认当前目录）</td>
</tr>
</tbody>
</table>

运行以下命令仅生成加密密钥分区:

```bash
python nvs_partition_gen.py generate-key
```

生成 NVS 加密分区 使用方法:

```bash
python nvs_partition_gen.py encrypt [-h] [--version (1,2)] [-keygen] [--keyfile KEYFILE] [-inputkey INPUTKEY] [-outdir OUTDIR] input output size
```

位置参数:

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>待解析 CSV 文件的路径</td>
</tr>
<tr>
<td>output</td>
<td>NVS 二进制文件的输出路径</td>
</tr>
<tr>
<td>size</td>
<td>NVS 分区大小（以字节为单位，且为 4096 的整数倍）</td>
</tr>
</tbody>
</table>

可选参数:
### 参数

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>显示帮助信息并退出</td>
</tr>
<tr>
<td>-version [1,2]</td>
<td>设置多页 Blob 版本。版本 1: 禁用多页 Blob;版本 2: 启用多页 Blob;默认版本: 版本 2。</td>
</tr>
<tr>
<td>--keygen</td>
<td>生成 NVS 分区加密密钥</td>
</tr>
<tr>
<td>--keyfile KEYFILE</td>
<td>密钥文件的输出路径</td>
</tr>
<tr>
<td>--inputkey INPUTKEY</td>
<td>内含 NVS 分区加密密钥的文件</td>
</tr>
<tr>
<td>--outdir OUTDIR</td>
<td>输出目录，用于存储创建的文件（默认当前目录）</td>
</tr>
</tbody>
</table>

运行以下命令加密 NVS 分区，该程序同时会提供一个 CSV 示例文件。

- 通过 NVS 分区生成程序生成加密密钥来加密:

```bash
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin
--0x3000 --keygen
```

**备注:** 创建的加密密钥格式为 `<outdir>/keys/keys-<timestamp>.bin`。

- 通过 NVS 分区生成程序生成加密密钥、并将密钥存储于自定义的文件中:

```bash
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin
--0x3000 --keygen --keyfile sample_keys.bin
```

**备注:** 创建的加密密钥格式为 `<outdir>/keys/keys-<timestamp>.bin`。

- 将加密密钥用作二进制输入文件来进行加密:

```bash
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin
--0x3000 --inputkey sample_keys.bin
```

### 解密 NVS 分区 使用方法:

```bash
python nvs_partition_gen.py decrypt [-h] [--outdir OUTDIR] input key output
```

**位置参数:**

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>待解析的 NVS 加密分区文件路径</td>
</tr>
<tr>
<td>key</td>
<td>含有解密密钥的文件路径</td>
</tr>
<tr>
<td>output</td>
<td>已解密的二进制文件输出路径</td>
</tr>
</tbody>
</table>

**可选参数:**

<table>
<thead>
<tr>
<th>参数</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>显示帮助信息并退出</td>
</tr>
<tr>
<td>--outdir OUTDIR</td>
<td>输出目录，用于存储创建的文件（默认当前目录）</td>
</tr>
</tbody>
</table>
运行以下命令解密已加密的 NVS 分区:

```
python nvs_partition_gen.py decrypt sample_encr.bin sample_keys.bin sample_decr.bin
```

您可以自定义格式版本号：- 版本 1: 禁用多 页 Blob - 版本 2: 启用多 页 Blob

**版本 1: 禁用多 页 Blob** 如需禁用多 页 Blob，请按照如下命令将版本参数设置为 1，以此格式运行分区生成程序。该程序同时会提供一个 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_singlepage_blob.csv sample.bin 0x3000 --version 1
```

**版本 2: 启用多 页 Blob** 如需启用多 页 Blob，请按照如下命令将版本参数设置为 2，以此格式运行分区生成程序。该程序同时会提供一个 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_multipage_blob.csv sample.bin 0x4000 --version 2
```

备注：NVS 分区最小为 0x3000 字节。

备注：将二进制文件烧录至设备时，请确保与应用的 sdkconfig 设置一致。

说明

- 分区生成程序不会对重复键进行检查，而将数据同时写入这两个重复键中。请注意不要使用同名的键。
- 新页面创建后，前一页的空白处不会再写入数据。CSV 文件中的字段须按次序排列以优化内存；
- 暂不支持 64 位数据类型。

### 2.9.5 NVS 分区解析程序

**介绍**

NVS 分区解析程序 nvs_flash/nvs_partition_parser/nvs_read.py 加载并解析 NVS 存储分区，以便于调试和数据提取。该程序还支持完整性检查功能，可扫描分区中可能存在的错误。

**加密分区**

此程序不支持解密。如需解密 NVS 分区，请使用 NVS 分区生成程序。该工具支持 NVS 分区加密。

**使用方法**

该程序通过 -d 或 -dump 选项提供了六种不同的输出方式：
- *all* (默认) - 打印所有含有元数据的条目。
- *written* - 只打印含有元数据的写入条目。
- *minimal* - 打印写入的 namespace: key = value 对。
- *namespaces* - 打印所有写入的命名空间。
- *blobs* - 打印所有 blob 和字符串（若 blob 和字符串是以分块的形式，则对其进行重组）。
• storage_info - 打印每一页的条目状态计数。

注意：该程序还提供 none 选项，该选项不会打印任何东西。如果 NVS 分区的内容并不相关，可以将该选项和完整性检查选项一起使用。

该程序支持完整性检查功能，选择选项 -i 或 --integrity-check 即可运行。此功能可扫描整个分区，并打印出可能存在的错误。当此功能和 -d none 一起使用时，可只打印可能存在的错误。

### 2.9.6 SD/SDIO/MMC 驱动程序

概述

SD/SDIO/MMC 驱动是一种基于 SDMMC 和 SD SPI 主机驱动的协议级驱动程序，目前已支持 SD 存储器、SDIO 卡和 eMMC 芯片。

SDMMC 主机驱动和 SD SPI 主机驱动（driver/include/driver/sdmmc_host.h 和 driver/include/driver/sdspi_host.h）为以下功能提供 API:

- 发送命令至从设备
- 接收和发送数据
- 处理总线错误

初始化函数及配置函数:

- 如需初始化和配置 SDMMC 主机，请参阅 SDMMC 主机 API
- 如需初始化和配置 SD SPI 主机，请参阅 SD SPI 主机 API

本文档中所述的 SDMMC 协议层仅处理 SD 协议相关事项，例如卡初始化和数据传输命令。

协议层通过 sdmmc_host_t 结构体和主机协同工作，该结构体包含指向主机各类函数的指针。

应用示例

ESP-IDF storage/sd_card 目录下提供了 SDMMC 驱动与 FatFs 库组合使用的示例，演示了先初始化卡，然后使用 POSIX 和 C 库 API 向卡读写数据。请参考示例目录下 README.md 文件，查看更多详细信息。

复合法（存储 + IO） 该驱动程序不支持 SD 复合法，复杂卡会被视为 IO 卡。

线程安全 多数应用程序仅需在一个任务中使用协议层。因此，协议层在 sdmmc_card_t 结构体或在访问 SDMMC 或 SD SPI 主机驱动程序时不使用任何类型的锁。这种锁通常在较高层级实现，例如文件系统驱动程序。

协议层 API

协议层具备 sdmmc_host_t 结构体，此结构体描述了 SD/MMC 主机驱动，列出了其功能，并提供指向驱动程序函数的指针。协议层将卡信息储存于 sdmmc_card_t 结构体中，向 SD/MMC 主机发送命令时，协议层使用 sdmmc_command_t 结构体来描述命令、参数、预期返回值和需传输的数据 (如有)。
用于 SD 存储卡的 API

1. 初始化主机，调用主机驱动函数，例如 `sdmmc_host_init()` 和 `sdmmc_host_init_slot()`；
2. 初始化卡，调用 `sdmmc_card_init()`，并将参数 host（主机驱动信息）和参数 card（指向 `sdmmc_card_t` 结构体的指针）传递给此函数。函数运行结束后，将会向 `sdmmc_card_t` 结构体填充该卡的信息；
3. 读取或写入卡的数据，请分别调用 `sdmmc_read_sectors()` 和 `sdmmc_write_sectors()`，并将参数 card（指向卡信息结构的指针）传递给函数；
4. 如果不再使用该卡，请调用主机驱动函数，例如 `sdmmc_host_deinit()`，以禁用主机外设，并释放驱动程序分配的资源。

用于 eMMC 芯片的 API 从协议层的角度而言，eMMC 存储芯片与 SD 存储卡相同。尽管 eMMC 是芯片，不具备卡的外形，但由于协议相似(`sdmmc_card_t, sdmmc_card_init`)，用于 SD 卡的一些概念同样适用于 eMMC 芯片。注意，eMMC 芯片不可通过 SPI 使用，因此它与 SD API 主机驱动不兼容。

如需初始化 eMMC 内存并执行读/写操作，请参照上一章节 SD 卡操作步骤。

用于 SDIO 卡的 API 由于卡初始化和检测过程与 SD 存储卡相同，唯一的区别是 SDIO 模式下数据传输命令不同。

在卡初始化和卡检测（通过运行 `sdmmc_card_init()`）期间，驱动仅配置 IO 卡如下寄存器：

1. I/O 中止 (0x06) 寄存器：在该寄存器中设置 RES（位可重置卡的 IO 部分）
2. 总线接口控制 (0x07) 寄存器：如果主机和插槽配置中启用 4 线模式，则驱动程序会尝试在该寄存器中设置总线宽度字段。如果字段设置成功，则从机支持 4 线模式，主机也切换至 4 线模式；
3. 高速 (0x13) 寄存器：如果主机配置中启用高速模式，则该寄存器的 HS 位会被设置。

注意，驱动程序不会在 (1) I/O 使能寄存器和 Int 使能寄存器，及 (2) I/O 块大小中，设置任何位。应用程序可通过调用 `sdmmc_io_read_byte()` 来设置相关位。

如需卡配置或传输数据，请根据您的具体情况选择下表中的函数：

<table>
<thead>
<tr>
<th>操作</th>
<th>函数读取</th>
<th>函数写入</th>
</tr>
</thead>
<tbody>
<tr>
<td>使用 IO_RW_DIRECT (CMD52) 读写单个字节。</td>
<td><code>sdmmc_io_read_byte()</code></td>
<td><code>sdmmc_io_write_byte()</code></td>
</tr>
<tr>
<td>使用 IO_RW_EXTENDED (CMD53) 读写多个字节。</td>
<td><code>sdmmc_io_read_bytes()</code></td>
<td><code>sdmmc_io_write_bytes()</code></td>
</tr>
<tr>
<td>块模式下，使用 IO_RW_EXTENDED (CMD53) 读写数据</td>
<td><code>sdmmc_io_read_blocks()</code></td>
<td><code>sdmmc_io_write_blocks()</code></td>
</tr>
</tbody>
</table>

使用 `sdmmc_io_enable_int()` 函数，应用程序可启用 SDIO 中断。在单线模式下使用 SDIO 时，还需要连接 D1 线来启用 SDIO 中断。

如果您需要应用程序保持等待直到发生 SDIO 中断，请使用 `sdmmc_io_wait_int()` 函数。

如果您您与 ESP32 的 SDIO 从设备通信，请使用 ESSL 组件 (ESP 串行从设备链接)，请参阅 `ESP Serial Slave Link` 和 peripherals/sdio/host。

**API 参考**

**Header File**
- components/sdmmc/include/sdmmc_cmd.h

**Functions**

```c
esp_err_t sdmmc_card_init (const sdmmc_host_t *host, sdmmc_card_t *out_card)
```

Probe and initialize SD/MMC card using given host

Espressif Systems 1678 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
Chapter 2. API reference

备注: Only SD cards (SDSC and SDHC/SDXC) are supported now. Support for MMC/eMMC cards will be added later.

参数
- host - pointer to structure defining host controller
- out_card - pointer to structure which will receive information about the card when the function completes

返回
- ESP_OK on success
- One of the error codes from SDMMC host controller

void sdmmc_card_print_info (FILE *stream, const sdmmc_card_t *card)
Print information about the card to a stream.

参数
- stream - stream obtained using fopen or fdopen
- card - card information structure initialized using sdmmc_card_init

esp_err_t sdmmc_get_status (sdmmc_card_t *card)
Get status of SD/MMC card

参数 card - pointer to card information structure previously initialized using sdmmc_card_init
返回
- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t sdmmc_write_sectors (sdmmc_card_t *card, const void *src, size_t start_sector, size_t sector_count)
Write given number of sectors to SD/MMC card

参数
- card - pointer to card information structure previously initialized using sdmmc_card_init
- src - pointer to data buffer to read data from; data size must be equal to sector_count * card->csd.sector_size
- start_sector - sector where to start writing
- sector_count - number of sectors to write

返回
- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t sdmmc_read_sectors (sdmmc_card_t *card, void *dst, size_t start_sector, size_t sector_count)
Read given number of sectors from the SD/MMC card

参数
- card - pointer to card information structure previously initialized using sdmmc_card_init
- dst - pointer to data buffer to write into; buffer size must be at least sector_count * card->csd.sector_size
- start_sector - sector where to start reading
- sector_count - number of sectors to read

返回
- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t sdmmc_erase_sectors (sdmmc_card_t *card, size_t start_sector, size_t sector_count, sdmmc_erase_arg_t arg)
Erase given number of sectors from the SD/MMC card

备注: When sdmmc_erase_sectors used with cards in SDSPI mode, it was observed that card requires re-init
after erase operation.

### 参数
- card – 指向先前初始化的SD/MMC卡信息结构
- start_sector – 要开始擦除的扇区
- sector_count – 要擦除的扇区数量
- arg – 擦除命令(CMD38)的参数

### 返回
- ESP_OK表示成功
- 其中一个错误代码来自SDMMC主机控制器

```c
esp_err_t sdmmc_can_discard(sdmmc_card_t *card)
```
检查SD/MMC卡是否支持Discard

- 参数 card – 指向先前初始化的SD/MMC卡信息结构

### 返回
- ESP_OK表示受支持的卡/设备
- ESP_FAIL表示不受支持的卡/设备

```c
esp_err_t sdmmc_can_trim(sdmmc_card_t *card)
```
检查SD/MMC卡是否支持Trim

- 参数 card – 指向先前初始化的SD/MMC卡信息结构

### 返回
- ESP_OK表示受支持的卡/设备
- ESP_FAIL表示不受支持的卡/设备

```c
esp_err_t sdmmc_mmc_can_sanitize(sdmmc_card_t *card)
```
检查SD/MMC卡是否支持Sanitize

- 参数 card – 指向先前初始化的SD/MMC卡信息结构

### 返回
- ESP_OK表示受支持的卡/设备
- ESP_FAIL表示不受支持的卡/设备

```c
esp_err_t sdmmc_mmc_sanitize(sdmmc_card_t *card, uint32_t timeout_ms)
```
Sanitize未映射的数据

- 参数 card – 指向先前初始化的SD/MMC卡信息结构
- timeout_ms – 擦除选定范围所需的时间

### 返回
- ESP_OK表示成功
- 其中一个错误代码来自SDMMC主机控制器

```c
esp_err_t sdmmc_full_erase(sdmmc_card_t *card)
```
Erase完成SD/MMC卡

- 参数 card – 指向先前初始化的SD/MMC卡信息结构

### 返回
- ESP_OK表示成功
- 其中一个错误代码来自SDMMC主机控制器
**esp_err_t sdmmc_io_read_byte (sdmmc_card_t *card, uint32_t function, uint32_t reg, uint8_t *out_byte)**

Read one byte from an SDIO card using IO_RW_DIRECT (CMD52)

**参数**
- card - pointer to card information structure previously initialized using sdmmc_card_init
- function - IO function number
- reg - byte address within IO function
- out_byte - [out] output, receives the value read from the card

**返回**
- ESP_OK on success
- One of the error codes from SDMMC host controller

**esp_err_t sdmmc_io_write_byte (sdmmc_card_t *card, uint32_t function, uint32_t reg, uint8_t in_byte, uint8_t *out_byte)**

Write one byte to an SDIO card using IO_RW_DIRECT (CMD52)

**参数**
- card - pointer to card information structure previously initialized using sdmmc_card_init
- function - IO function number
- reg - byte address within IO function
- in_byte - value to be written
- out_byte - [out] if not NULL, receives new byte value read from the card (read-after-write).

**返回**
- ESP_OK on success
- One of the error codes from SDMMC host controller

**esp_err_t sdmmc_io_read_bytes (sdmmc_card_t *card, uint32_t function, uint32_t addr, void *dst, size_t size)**

Read multiple bytes from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in byte mode. For block mode, see sdmmc_io_read_blocks.

**参数**
- card - pointer to card information structure previously initialized using sdmmc_card_init
- function - IO function number
- addr - byte address within IO function where reading starts
- dst - buffer which receives the data read from card
- size - number of bytes to read

**返回**
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

**esp_err_t sdmmc_io_write_bytes (sdmmc_card_t *card, uint32_t function, uint32_t addr, const void *src, size_t size)**

Write multiple bytes to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in byte mode. For block mode, see sdmmc_io_write_blocks.

**参数**
- card - pointer to card information structure previously initialized using sdmmc_card_init
- function - IO function number
- addr - byte address within IO function where writing starts
- src - data to be written
- size - number of bytes to write

**返回**
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
Chapter 2: API

- One of the error codes from SDMMC host controller

**esp_err_t sdmmc_io_read_blocks (sdmmc_card_t *card, uint32_t function, uint32_t addr, void *dst, size_t size)**

Read blocks of data from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in block mode. For byte mode, see sdmmc_io_read_bytes.

**参数**
- **card** - pointer to card information structure previously initialized using sdmmc_card_init
- **function** - IO function number
- **addr** - byte address within IO function where writing starts
- **dst** - buffer which receives the data read from card
- **size** - number of bytes to read, must be divisible by the card block size.

**返回**
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

**esp_err_t sdmmc_io_write_blocks (sdmmc_card_t *card, uint32_t function, uint32_t addr, const void *src, size_t size)**

Write blocks of data to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in block mode. For byte mode, see sdmmc_io_write_bytes.

**参数**
- **card** - pointer to card information structure previously initialized using sdmmc_card_init
- **function** - IO function number
- **addr** - byte address within IO function where writing starts
- **src** - data to be written
- **size** - number of bytes to read, must be divisible by the card block size.

**返回**
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

**esp_err_t sdmmc_io_enable_int (sdmmc_card_t *card)**

Enable SDIO interrupt in the SDMMC host

**参数**
- **card** - pointer to card information structure previously initialized using sdmmc_card_init

**返回**
- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts

**esp_err_t sdmmc_io_wait_int (sdmmc_card_t *card, TickType_t timeout_ticks)**

Block until an SDIO interrupt is received

Slave uses D1 line to signal interrupt condition to the host. This function can be used to wait for the interrupt.

**参数**
- **card** - pointer to card information structure previously initialized using sdmmc_card_init
- **timeout_ticks** - time to wait for the interrupt, in RTOS ticks

**返回**
- ESP_OK if the interrupt is received
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts
- ESP_ERR_TIMEOUT if the interrupt does not happen in timeout_ticks

**esp_err_t sdmmc_io_get_cis_data (sdmmc_card_t *card, uint8_t *out_buffer, size_t buffer_size, size_t *inout_cis_size)**

Get the data of CIS region of an SDIO card.
You may provide a buffer not sufficient to store all the CIS data. In this case, this function stores as much data into your buffer as possible. Also, this function will try to get and return the size required for you.

### Parameters
- `card` - Pointer to card information structure previously initialized using `sdmmc_card_init`
- `out_buffer` - Output buffer of the CIS data
- `buffer_size` - Size of the buffer.
- `inout_cis_size` - Mandatory, pointer to a size, input and output.
  - input: Limitation of maximum searching range, should be 0 or larger than `buffer_size`.
    The function searches for `CIS_CODE_END` until this range. Set to 0 to search infinitely.
  - output: The size required to store all the CIS data, if `CIS_CODE_END` is found.

### Return
- `ESP_OK`: on success
- `ESP_ERR_INVALID_RESPONSE`: if the card does not (correctly) support CIS.
- `ESP_ERR_INVALID_SIZE`: `CIS_CODE_END` found, but `buffer_size` is less than required size, which is stored in the `inout_cis_size` then.
- `ESP_ERR_NOT_FOUND`: if the `CIS_CODE_END` not found. Increase input value of `inout_cis_size` or set it to 0, if you still want to search for the end; output value of `inout_cis_size` is invalid in this case.
- and other error code return from `sdmmc_io_read_bytes`

```c
esp_err_t sdmmc_io_print_cis_info (uint8_t *buffer, size_t buffer_size, FILE *fp)
```

### Parse and print the CIS information of an SDIO card.

### Notes
Not all the CIS codes and all kinds of tuples are supported. If you see some unresolved code, you can add the parsing of these code in `sdmmc_io.c` and contribute to the IDF through the Github repository.

```c
using sdmmc_card_init
```

---

### Parameters
- `buffer` - Buffer to parse
- `buffer_size` - Size of the buffer.
- `fp` - File pointer to print to, set to NULL to print to stdout.

### Return
- `ESP_OK`: on success
- `ESP_ERR_NOT_SUPPORTED`: if the value from the card is not supported to be parsed.
- `ESP_ERR_INVALID_SIZE`: if the CIS size fields are not correct.

### Header File
- `components/driver/include/driver/sdmmc_types.h`

### Structures

```c
struct sdmmc_csd_t
```
Decoded values from SD card Card Specific Data register

### Public Members

```c
int csd_ver
```
CSD structure format
int mmc_ver
  MMC version (for CID format)

int capacity
  total number of sectors

int sector_size
  sector size in bytes

int read_block_len
  block length for reads

int card_command_class
  Card Command Class for SD

int tr_speed
  Max transfer speed

struct sdmmc_cid_t
  Decoded values from SD card Card IDentification register

Public Members

int mfg_id
  manufacturer identification number

int oem_id
  OEM/product identification number

char name[8]
  product name (MMC v1 has the longest)

int revision
  product revision

int serial
  product serial number

int date
  manufacturing date

struct sdmmc_scr_t
  Decoded values from SD Configuration Register Note: When new member is added, update reserved bits accordingly

Public Members
Chapter 2. API 参考

- `uint32_t sd_spec`
  SD Physical layer specification version, reported by card

- `uint32_t erase_mem_state`
  Data state on card after erase whether 0 or 1 (card vendor dependent)

- `uint32_t bus_width`
  Bus widths supported by card: BIT(0) — 1-bit bus, BIT(2) — 4-bit bus

- `uint32_t reserved`
  Reserved for future expansion

- `uint32_t rsvd_mnf`
  Reserved for manufacturer usage

**struct sdmmc_ssr_t**
Decoded values from SD Status Register Note: When new member is added, update reserved bits accordingly

**Public Members**

- `uint32_t alloc_unit_kb`
  Allocation unit of the card, in multiples of kB (1024 bytes)

- `uint32_t erase_size_au`
  Erase size for the purpose of timeout calculation, in multiples of allocation unit

- `uint32_t cur_bus_width`
  SD current bus width

- `uint32_t discard_support`
  SD discard feature support

- `uint32_t fule_support`
  SD FULE (Full User Area Logical Erase) feature support

- `uint32_t erase_timeout`
  Timeout (in seconds) for erase of a single allocation unit

- `uint32_t erase_offset`
  Constant timeout offset (in seconds) for any erase operation

- `uint32_t reserved`
  Reserved for future expansion

**struct sdmmc_ext_csd_t**
Decoded values of Extended Card Specific Data
Public Members

`uint8_t rev`
Extended CSD Revision

`uint8_t power_class`
Power class used by the card

`uint8_t erase_mem_state`
data state on card after erase whether 0 or 1 (card vendor dependent)

`uint8_t sec_feature`
secure data management features supported by the card

`struct sdmmc_switch_func_rsp_t`
SD SWITCH_FUNC response buffer

Public Members

`uint32_t data[512 / 8 / sizeof(uint32_t)]`
response data

`struct sdmmc_command_t`
SD/MMC command information

Public Members

`uint32_t opcode`
SD or MMC command index

`uint32_t arg`
SD/MMC command argument

`sdmmc_response_t response`
response buffer

`void *data`
buffer to send or read into

`size_tdatalen`
length of data buffer

`size_tblklen`
block length

`int flags`
see below
**Chapter 2. API**


eesp_err_t error

error returned from transfer

uint32_t timeout_ms

response timeout, in milliseconds

struct sdmmc_host_t

SD/MMC Host description

This structure defines properties of SD/MMC host and functions of SD/MMC host which can be used by upper layers.

**Public Members**

uint32_t flags

flags defining host properties

int slot

slot number, to be passed to host functions

int max_freq_khz

max frequency supported by the host

float io_voltage

I/O voltage used by the controller (voltage switching is not supported)

eesp_err_t (*init)(void)

Host function to initialize the driver

eesp_err_t (*set_bus_width)(int slot, size_t width)

host function to set bus width

size_t (*get_bus_width)(int slot)

host function to get bus width

eesp_err_t (*set_bus_ddr_mode)(int slot, bool ddr_enable)

host function to set DDR mode

eesp_err_t (*set_card_clk)(int slot, uint32_t freq_khz)

host function to set card clock frequency

eesp_err_t (*do_transaction)(int slot, sdmmc_command_t *cmdinfo)

host function to do a transaction

eesp_err_t (*deinit)(void)

host function to deinitialize the driver

eesp_err_t (*deinit_p)(int slot)

host function to deinitialize the driver, called with the slot
Chapter 2. API

**esp_err_t (*io_int_enable)(int slot)**
Host function to enable SDIO interrupt line

**esp_err_t (*io_int_wait)(int slot, TickType_t timeout_ticks)**
Host function to wait for SDIO interrupt line to be active

**int command_timeout_ms**
timeout, in milliseconds, of a single command. Set to 0 to use the default value.

**esp_err_t (*get_real_freq)(int slot, int *real_freq)**
Host function to provide real working freq, based on SDMMC controller setup

**struct sdmmc_card_t**
SD/MMC card information structure

### Public Members

**sdmmc_host_t host**
Host with which the card is associated

**uint32_t ocr**
OCR (Operation Conditions Register) value

**sdmmc_cid_t cid**
decoded CID (Card IDentification) register value

**sdmmc_response_t raw_cid**
raw CID of MMC card to be decoded after the CSD is fetched in the data transfer mode

**sdmmc_csd_t csd**
decoded CSD (Card-Specific Data) register value

**sdmmc_scr_t scr**
decoded SCR (SD card Configuration Register) value

**sdmmc_ssr_t ssr**
decoded SSR (SD Status Register) value

**sdmmc_ext_csd_t ext_csd**
decoded EXT_CSD (Extended Card Specific Data) register value

**uint16_t rca**
RCA (Relative Card Address)

**uint16_t max_freq_khz**
Maximum frequency, in kHz, supported by the card
int real_freq_khz
    Real working frequency, in kHz, configured on the host controller

uint32_t is_mem
    Bit indicates if the card is a memory card

uint32_t is_sdio
    Bit indicates if the card is an IO card

uint32_t is mmc
    Bit indicates if the card is MMC

uint32_t num_io_functions
    If is_sdio is 1, contains the number of IO functions on the card

uint32_t log bus width
    log2(bus width supported by card)

uint32_t is ddr
    Card supports DDR mode

uint32_t reserved
    Reserved for future expansion

Macros

SDMMC_HOST_FLAG_1BIT
    host supports 1-line SD and MMC protocol

SDMMC_HOST_FLAG_4BIT
    host supports 4-line SD and MMC protocol

SDMMC_HOST_FLAG_8BIT
    host supports 8-line MMC protocol

SDMMC_HOST_FLAG_SPI
    host supports SPI protocol

SDMMC_HOST_FLAG_DDR
    host supports DDR mode for SD/MMC

SDMMC_HOST_FLAG_DEINIT_ARG
    host deinit function called with the slot argument

SDMMC_FREQ_DEFAULT
    SD/MMC Default speed (limited by clock divider)

SDMMC_FREQ_HIGHSPEED
    SD High speed (limited by clock divider)
**Type Definitions**

```c
typedef uint32_t sdmmc_response_t[4]
```
SD/MMC command response buffer

**Enumerations**

```c
enum sdmmc_erase_arg_t
```
SD/MMC erase command (38) arguments

- **SDMMC_ERASE_ARG**
  - Erase operation on SD, Trim operation on MMC

- **SDMMC_DISCARD_ARG**
  - Discard operation for SD/MMC

### 2.9.7 SPI Flash API

**概述**

spi_flash 组件提供外部 flash 数据读取、写入、擦除和内存映射相关的 API 函数，同时也提供了更高层的、面向分区的 API 函数（定义在 esp API 中）。

与 ESP-IDF V4.0 之前的 API 不同，这一版 esp_* API 功能并不局限于主 SPI flash 芯片（即运行程序的 SPI flash 芯片），使用不同的芯片指令，您可以访问连接到 SPI0/1 或 SPI2 总线的外部 flash 芯片。

**备注**：大多数 esp_* API 使用 SPI1，SPI2 等外设而非通过 SPI0 上的 cache。这使得它们不仅访问主 flash，也能访问外部 flash。
而由于 cache 的限制，所有经过 cache 的操作都只能对主 flash 进行。这些操作的地址同样受到 cache 能力的限制。Cache 无法访问外部 flash 或者高于它能力的地址段。这些 cache 操作包括：mmap，加密读写，执行代码或者访问在 flash 中的变量。

备注：ESP-IDF V4.0 之后的 flash API 不再是 原子 的。因此，如果读操作执行过程中发生写操作，且读操作和写操作的 flash 地址出现重叠，读操作返回的数据可能会包含旧数据和新数据（新数据为写操作更新产生的数据）。

备注：仅有主 flash 芯片支持加密操作，外接（经 SPI 使用其他不同片选访问，或经其它 SPI 总线访问）的 flash 芯片则不支持加密操作。硬件的限制也决定了仅有主 flash 支持从 cache 当中读取。

Flash 功能支持情况

支持的 Flash 列表 不同厂家的 flash 特性有不同的操作方式，因此需要特殊的驱动支持。当前驱动支持大多数厂家 flash 24 位地址范围内的快速/慢速读，以及二线模式 (DIO/DOUT)，因为它们不需要任何厂家的自定义命令。

当前驱动支持以下厂家/型号的 flash 的四线模式 (QIO/QOUT)：

1. ISSI
2. GD
3. MXIC
4. FM
5. Winbond
6. XMC
7. BOYA

Flash 可选的功能

Optional features for flash　Some features are not supported on all ESP chips and Flash chips. You can check the list below for more information.

• Auto Suspend & Resume
• Flash unique ID
• High performance mode
• OPI flash support
• 32-bit Address Flash Chips

备注：

• The features listed above needs to be supported by both esp chips and flash chips.
• If you are using an official Espressif modules/SiP. Some of the modules/SiPs always support the feature, in this case you can see these features listed in the datasheet. Otherwise please contact Espressif’s business team to know if we can supply such products for you.
• If you are making your own modules with your own bought flash chips, and you need features listed above. Please contact your vendor if they support the those features, and make sure that the chips can be supplied continuously.

注意：This document only shows that IDF code has supported the features of those flash chips. It’s not a list of stable flash chips certified by Espressif. If you build your own hardware from flash chips with your own brought flash chips (even with flash listed in this page), you need to validate the reliability of flash chips yourself.
Chapter 2. API 参考

**Auto Suspend & Resume**

ESP Chips List:

1. ESP32C3

Flash Chips List:

1. XM25QxxC series.

**Flash unique ID**

Unique ID is not flash id, which means flash has 64-Bit unique ID for each device. The instruction to read the unique ID (4Bh) accesses a factory-set read-only 64-bit number that is unique to each flash device. This ID number helps you to recognize each single device. Not all flash vendors support this feature. If you try to read the unique ID on a chip which does not have this feature, the behavior is not determined. The support list is as follows.

ESP Chips Lists:

ALL

Flash Chips List:

1. ISSI
2. GD
3. TH
4. FM
5. Winbond
6. XMC
7. BOYA

**High performance mode**

备注: This section is provided for Dual mode (DOUT/DIO) and Quad mode (QIO/QOUT) flash chips. Octal flash used on ESP-chips support High performance mode by default so far, you can refer to the octal flash support list below.

High performance mode (HPM) means that the SPI1 and flash chip works under high frequency. Usually, when the operating frequency of the flash is greater than 80MHz, it is considered that the flash works under HPM. As far as we acknowledged, flash chips have more than two different coping strategies when flash work under HPM. For some flash chips, HPM is controlled by high performance flag (HPF) in status register and for some flash chips, HPM is controlled by dummy cycle bit.

For following conditions, IDF start code deals with HPM internally.

ESP Chips List:

1. ESP32S3

Flash Chips (name & ID) List:

1. GD25Q64C (ID: 0xC84017)
2. GD25Q32C (ID: 0xC84016)

**OPI flash support**

OPI flash means that the flash chip supports octal peripheral interface, which has octal I/O pins. Different octal flash has different configurations and different commands. Hence, it is necessary to carefully check the support list.

ESP Chips List:

1. ESP32S3
Chapter 2. API 参考

Flash Chips List:

1. MX25UM25645G

**32-bit Address Flash Chips**   Most NOR flash chips used by Espressif chips use 24-bits address, which can cover 16 MBBytes memory. However, for larger memory (usually equal to or larger than 16 MBBytes), flash uses a 32-bits address to address larger memory. Regretfully, 32-bits address chips have vendor-specific commands, so we need to support the chips one by one.

ESP Chips List:

ALL ESP Chips support this.

Flash Chips List:

1. W25Q256
2. GD25Q256

有一些功能可能不是所有的 flash 芯片都支持，或不是所有的 ESP 芯片都支持。这些功能包括：

- 32 比特地址的 flash 支持 - 通常意味着拥有大于 16MB 内存空间的大容量 flash 需要更长的地址去访问。

- flash 的私有 ID (unique ID) - 表示 flash 支持它自己的 64-bits 独有 ID。

如果您想使用这些功能，则需保证 ESP32 支持这些功能，且产品里所使用的 flash 芯片也要支持这些功能。请参阅 Optional features for flash，查看更多信息。

您也可以自定义 flash 芯片驱动。请参阅 Overriding Default Chip Drivers，查看详细信息。

警告: Customizing SPI Flash Chip Drivers is considered an "expert" feature. Users should only do so at their own risk. (See the notes below)

**Overriding Default Chip Drivers**   During the SPI Flash driver’s initialization (i.e., esp_flash_init()), there is a chip detection step during which the driver will iterate through a Default Chip Driver List and determine which chip driver can properly support the currently connected flash chip. The Default Chip Drivers are provided by the IDF, thus are updated in together with each IDF version. However IDF also allows users to customize their own chip drivers.

Users should note the following when customizing chip drivers:

1. You may need to rely on some non-public IDF functions, which have slight possibility to change between IDF versions. On the one hand, these changes may be useful bug fixes for your driver, on the other hand, they may also be breaking changes (i.e., breaks your code).
2. Some IDF bug fixes to other chip drivers will not be automatically applied to your own custom chip drivers.
3. If the protection of flash is not handled properly, there may be some random reliability issues.
4. If you update to a newer IDF version that has support for more chips, you will have to manually add those new chip drivers into your custom chip driver list. Otherwise the driver will only search for the drivers in custom list you provided.

**Steps For Creating Custom Chip Drivers and Overriding the IDF Default Driver List**

1. Enable the `CONFIG_SPI_FLASH_OVERRIDE_CHIP_DRIVER_LIST` config option. This will prevent compilation and linking of the Default Chip Driver List (default_registered_chips) provided by IDF. Instead, the linker will search for the structure of the same name (default_registered_chips) that must be provided by the user.
2. Add a new component in your project, e.g. custom_chip_driver.
3. Copy the necessary chip driver files from the spi_flash component in IDF. This may include:
   - `spi_flash_chip_drivers.c` (to provide the default_registered_chips structure)
   - Any of the `spi_flash_chip_*.c` files that matches your own flash model best
   - `CMakeLists.txt` and `linker.lf` files
   Modify the files above properly.
Chapter 2. API

When writing your own flash chip driver, you can set your flash chip capabilities through `spi_flash_chip_***get_caps` and points the function pointer `get_chip_caps` for protection to the `spi_flash_chip_***get_caps` function. The steps are as follows.

1. Please check whether your flash chip have the capabilities listed in `spi_flash_caps_t` by checking the flash datasheet.
2. Write a function named `spi_flash_chip_***get_caps`. Take the example below as a reference. (if the flash support suspend and read unique id).
3. Points the pointer `get_chip_caps` (in `spi_flash_chip_t`) to the function mentioned above.

```c
spi_flash_caps_t spi_flash_chip_***get_caps(esp_flash_t *chip)
{
    spi_flash_caps_t caps_flags = 0;
    // 32-bit-address flash is not supported
    flash-suspend is supported
    caps_flags |= SPI_FLASH_CHIP_CAP_SUSPEND;
    // flash read unique id.
    caps_flags |= SPI_FLASH_CHIP_CAP_UNIQUE_ID;
    return caps_flags;
}
```

4. Add linking dependency from `spi_flash` component to the new `custom_chip_driver` component, by adding the following lines after the `idf_component_register`, in the `CMakeLists.txt` file of the `custom_chip_driver` component:

```c
idf_component_get_property(spi_flash_lib spi_flash COMPONENT_LIB)
set_property(TARGET ${spi_flash_lib} APPEND PROPERTY INTERFACE_LINK_LIBRARIES$<LINK_ONLY:${COMPONENT_LIB}>)
```

5. The `linker.lf` is used to put every chip driver that you are going to use whilst cache is disabled into internal RAM. See more details. Make sure this file covers all the source files that you add.

6. Build your project, and you will see the new flash driver is used.

**Example** See also `storage/custom_flash_driver`.

### 初始化 Flash 设备

在使用 `esp_flash_*` API 之前，您需要在 SPI 总线上初始化芯片，步骤如下：

1. 调用 `spi_bus_initialize()` 初始化 SPI 总线。此函数将初始化总线上设备间共享的资源，如 I/O、DMA、中断等。
2. 调用 `spi_bus_add_flash_device()` 将 flash 设备连接到总线上。然后分配内存，填充 `esp_flash_t` 结构体，同时初始化 CS I/O。
3. 调用 `esp_flash_init()` 与芯片进行通信。后续操作会依据芯片类型不同而有差异。

**备注：** 当前，已支持多个 flash 芯片连接到同一总线。

### SPI Flash 访问 API

如下所示为处理 flash 中数据的函数集：
Chapter 2. API 参考

- `esp_flash_read()`：将数据从 flash 读取到 RAM；
- `esp_flash_write()`：将数据从 RAM 写入到 flash；
- `esp_flash_erase_region()`：擦除 flash 中指定区域的数据；
- `esp_flash_erase_chip()`：擦除整个 flash；
- `esp_flash_get_chip_size()`：返回 menuconfig 中设置的 flash 芯片容量（以字节为单位）。

一般来说，请尽量避免对 SPI flash 芯片直接使用原始 SPI flash 函数。如需对 SPI flash 芯片进行操作，请使用分区专用函数。

SPI Flash 容量

SPI flash 容量由引导加载程序镜像头部（烧录偏移为 0x1000）的一个字段进行配置。

默认情况下，引导程序被写入 flash 时，esptool.py 会自动检测 SPI flash 容量，同时使用正确容量更新引导程序的头部。您也可以在工程配置中设置 `CONFIG_ESPTOOLPY_FLASHSIZE`，生成固定的 flash 容量。

如需在运行时覆盖已配置的 flash 容量，请配置 g_rom_flashchip 结构中的 chip_size。`esp_flash_*` 函数使用此容量（于软件和 ROM 中）进行边界检查。

SPII Flash 并发约束

SPII flash 并发约束 指令/数据 cache（用以执行固件）与 SPII 外设（由像 SPI flash 驱动一样的驱动程序控制）共享 SPII/1 总线。因此，SPII 外设上的操作会对整个系统造成显著的影响。这类操作包括调用 SPI flash API 或者 SPII 总线上的其他驱动。任何 flash 操作（如读取、写入、擦除）或是由其他用户定义的 SPI 操作（对主 flash 或是其他 SPI 从机）。

在 ESP32 上，flash 读取/写入/擦除时，cache 必须被禁用。

当 cache 被禁用时 此时，在 flash 擦写操作中，所有的 CPU 都只能执行 IRAM 中的代码，而且必须从 DRAM 中读取数据。如果您使用本文档中 API 函数，上述限制将自动生效且透明（无需您额外关注），但这些限制可能影响系统中其他任务的性能。

要避免意外读取 flash cache，一个 CPU 在启动 flash 写入或擦除操作时，另一个 CPU 将阻塞。在 flash 操作完成前，所有 CPU 上，所有的非 IRAM 安全的中断都会被禁用。

另请参阅 OS 函数和 SPI 总线。

除 SPII/1 以外，SPI 总线上的其他 flash 芯片不受这种限制。

请参阅应用程序内存分布，查看内部 RAM（如 IRAM、DRAM）和 flash cache 的区别。

IRAM 安全中断处理程序 如果您需要在 flash 操作期间运行中断处理程序（比如低延迟操作），请在 注册中断处理程序 时设置 `ESP_INTR_FLAG_IRAM`。

请确保中断处理程序访问的所有数据和函数（包括其调用的数据和函数）都存储在 IRAM 或 DRAM 中。参见如何将代码放入 IRAM。

在函数或符号未被正确放入 IRAM/DRAM 的情况下，中断处理程序在 flash 操作期间从 flash cache 中读取数据时，会导致程序崩溃。这可能是因为代码未被正确放入 IRAM 而产生非法指令异常，也可能是因为常数未被正确放入 DRAM 而读取到垃圾数据。

备注：在 ISR 中处理字符串时，不建议使用 printf 和其他输出函数。为了方便调试，在从 ISRs 中获取数据时，请使用 ESP_DRAM_LOGE() 和类似的宏。请确保 TAG 和格式字符串都放置于 DRAM 中。
Chapter 2. API 参考

非 IRAM 安全中断处理程序 如果在注册时没有设置 ESP_INTR_FLAG_IRAM 标志，当 cache 被禁用时，将不会执行中断处理程序。一旦 cache 恢复，非 IRAM 安全的中断将重新启用，中断处理程序随即再次正常运行。这意味着，只要 cache 被禁用，将不会发生相应的硬件事件。

注意：指令/数据 cache（用以执行固件）与 SPI 外设（由像 SPI flash 驱动一样的驱动程序控制）共享 SPI0/1 总线。因此，在 SPI1 总线上调用 SPI flash API（包括访问 flash）会对整个系统造成显著的影响。请参阅 SPI flash 并发约束，查看详细信息。

分区表 API

ESP-IDF 工程使用分区表保存 SPI flash 各区信息，包括引导程序、各种应用程序二进制文件、数据及文件系统等。请参阅 分区表，查看详细信息。

该组件在 esp_partition.h 中声明了一些 API 函数，用以枚举在分区表中找到的分区，并对这些分区执行操作：

- `esp_partition_find()`：在分区表中查找特定类型的条目，返回一个不透明迭代器；
- `esp_partition_get()`：返回一个结构体，描述给定迭代器的分区；
- `esp_partition_next()`：将迭代器移至下一个找到的分区；
- `esp_partition_iterator_release()`：释放 `esp_partition_find` 中返回的迭代器；
- `esp_partition_find_first()`：返回描述 `esp_partition_find` 中找到的第一个分区的结构；
- `esp_partition_read()`、`esp_partition_write()` 和 `esp_partition_erase_range()` 等同于 `esp_flash_read()`、`esp_flash_write()` 和 `esp_flash_erase_region()`，但在分区表内进行。

备注：请在应用程序代码中使用上述 `esp_partition_*` API 函数，而非低层级的 `esp_flash_*` API 函数。分区表 API 函数根据存储在分区表中的数据，进行边界检查并计算在 flash 中的正确偏移量。

SPI Flash 加密

您可以对 SPI flash 内容进行加密，并在硬件层对其进行透明解密。

请参阅 flash 加密，查看详细信息。

内存映射 API

ESP32 的内存硬件可以将 flash 部分区域映射到指令地址空间和数据地址空间。此映射仅用于读操作，不能通过写入 flash 映射的存储区域来改变 flash 中的内容。

Flash 在 64 KB 页进行映射。内存映射硬件既可将 flash 映射到数据地址空间，也能映射到指令地址空间。请查看技术参考手册，了解内存映射硬件的详细信息及有关限制。

请注意，有些页被用于将应用程序映射到内存中，因此实际可用的页会少于硬件提供的总数。

启用 Flash 加密 时，使用内存映射区域从 flash 读取数据是解密闪 的唯一方法，解密需在硬件层进行。

内存映射 API 在 spi_flash_mmap.h 和 esp_partition.h 中声明：

- `spi_flash_mmap()`：将 flash 物理地址区域映射到 CPU 指令空间或数据空间；
- `spi_flash_munmap()`：取消上述区域的映射；
- `esp_partition_mmap()`：将分区的一部分映射至 CPU 指令空间或数据空间；
- `spi_flash_mmap()` 和 `esp_partition_mmap()` 的区别如下：
- `spi_flash_mmap()`：需要给定一个 64 KB 对齐的物理地址；
- `esp_partition_mmap()`：给定分区内任意偏移量即可，此函数根据需要将返回的指针调整至指
向映射内存。

Espressif Systems 1696 Release v5.1-dev-2066-g7869f4e151
内存映射以页为单位，即使传递给 esp_partition_mmap 的是一个分区，分区外的数据也是也是可以被读取到的，不会受到分区边界的影响。

备注：由于 mmap 是由 cache 支持的，因此，mmap 也只能用在主 flash 上。

SPI Flash 实现

esp_flash_t 结构体包含芯片数据和该 API 的三个重要部分：
1. 主机驱动，为访问芯片提供硬件支持；
2. 芯片驱动，为不同芯片提供兼容性服务；
3. OS 函数，在不同阶段（一级或二级 Boot 或者应用程序阶段）为部分 OS 函数（如锁、延迟）提供支持。

主机驱动 主机驱动依赖 hal/include/hal 文件夹下 spi_flash_types.h 定义的 spi_flash_host_driver_t 接口。该接口提供了一些常用的函数，用于与芯片通信。
在 SPI HAL 文件中，有些函数是基于现有的 ESP32 memory-spi 来实现的，但是，由于 ESP32 的速度限制，HAL 层无法提供某些芯片的高级功能（所以这些命令根本不在 HAL 的文件中被实现）。
memapi_host_driver.h 和 .c 文件使用 HAL 提供的 common_command 函数实现上述命令的高速版本，并将所有它实现的及 HAL 函数封装为 spi_flash_host_driver_t 供上层调用。
您甚至可以仅通过 GPIO 来实现自己的主机驱动。只要实现了 spi_flash_host_driver_t 中所有函数，不管底层硬件是什么，esp_flash_API 都可以访问 flash。

芯片驱动 芯片驱动在 spi_flash_chip_driver.h 中进行定义，并将主机驱动提供的基本函数进行封装以供 API 层使用。
有些操作需在执行前先发送命令，或在执行后读取状态，因此有些芯片需要不同的命令或值以及通信方式。
generic chip 芯片代表了常见的 flash 芯片，其他芯片驱动可以在这种通用芯片的基础上进行开发。
芯片驱动依赖主机驱动。

OS 函数 OS 函数层目前支持访问Flash和延迟的方法。
锁（见 SPI 总线锁）用于解决同一 SPI 总线上的设备访问和 SPI flash 芯片访问之间的冲突。例如：
1. 经 SPI 总线访问 flash 芯片时，应当禁用 cache（平时用于获取代码和 PSRAM 数据）。
2. 经其他总线访问 flash 芯片时，应当使用 flash 上 SPI 主驱动器注册的 ISR 以避免冲突。
3. SPI 主驱动器上某些没有 CS 线或者 CS 线受软件（如 SDSPI）控制的设备需要在一段时间内独占总线。
延时则用于某些长时操作，需要主机处于等待状态或执行轮询。
顶层 API 将芯片驱动和 OS 函数封装成一个完整的组件，并提供参数检查。
使用 OS 函数还可以在一定程度上避免在擦除大块 flash 区域时出现看门狗超时的情况。在这段时间内，
CPU 将被 flash 擦除任务占用，从而阻止其他任务的执行，包括为看门狗定时器 (WDT) 供电的空闲任务。若已选中配置选项 CONFIG_ESP_TASK_WDT_PANIC，并且 flash 操作时间长于看门狗的超时时间，系统将重新启动。
不过，由于不同的 flash 芯片擦除时间不同，flash 驱动几乎无法兼容，很难完全规避超时的风险。因此，
您需要格外注意这一点。请遵照以下指南：
1. 建议启用 CONFIG_SPI_FLASH_YIELD_DURING_ERASE 选项，允许调度器在擦除 flash 时进行重新调度。
此外，还可以使用下列参数。
• 在 menuconfig 中增加 CONFIG_SPI_FLASH_ERASE_YIELD_TICKS 或减少 CONFIG_SPI_FLASH_ERASE_YIELD_DURATION_MS 的时间。
2. 注意，在进行长时间的 SPI flash 操作时，启用 `CONFIG_ESP_TASK_WDT_PANIC` 选项将会在超时不触发恐慌处理程序。不过，启动该选项也可以帮助处理应用程序中的意外异常，您可以根据实际情况决定是否启用这个选项。

3. 在开发过程中，请根据项目对擦除 flash 的具体要求和时间限制，谨慎进行 flash 操作。在配置 flash 擦除超时周期时，请在实际产品要求的基础上留出合理的冗余时间，从而提高产品的可靠性。

另请参考

- `OTA API` 提供了高层 API 用于更新存储在 flash 中的 app 固件。
- `NVS API` 提供了结构化 API 用于存储 SPI flash 中的碎片数据。

实现细节

必须确保操作期间，两个 CPU 均未从 flash 运行代码。实现细节如下：

- 单核模式下，SDK 在执行 flash 操作前将禁用中断或调度算法。
- 双核模式下，SDK 需确保两个 CPU 均未运行 flash 代码。

如果用 SPI flash API 在 CPU A (PRO 或 APP) 上应用，它使用 `esp_ipc_call` API 在 CPU B 上运行 `spi_flash_op_block_func` 函数。`esp_ipc_call` API 会在 CPU B 上唤醒一个高优先级任务，即运行 `spi_flash_op_block_func` 函数。运行该函数将禁用 CPU B 上的 `cache`，并使用 `s_flash_op_can_start` 旗帜来标志 `cache` 已禁用。然后，CPU A 上的任务也会禁用 `cache` 并继续执行 flash 操作。

执行 flash 操作时，CPU A 和 CPU B 仍然可以执行中断操作。默认中断代码均存储于 RAM 中，如果新添加了中断分配 API，则应添加一个标志位以请求在 flash 操作期间禁用该新分配的中断。

Flash 操作完成后，CPU A 上的函数将设置另一标志位，即 `s_flash_op_complete`，用以通知 CPU B 上的任务可以重新启用 `cache` 并释放 CPU。接着，CPU A 上的函数也重新启用 `cache`，并将控制权返还给调用者。

另外，所有 API 函数均受互斥量 `s_flash_op_mutex` 保护。

在单核环境中（启用 `CONFIG_FREERTOS_UNICORE`），您需要禁用上述两个 `cache` 以防发生 CPU 间通信。

**SPI Flash API 参考**

**Header File**

- `components/spi_flash/include/esp_flash_spi_init.h`

**Functions**

```c
esp_err_t spi_bus_add_flash_device (esp_flash_t **out_chip, const esp_flash_spi_device_config_t *config)
```

Add a SPI Flash device onto the SPI bus.

The bus should be already initialized by `spi_bus_initialization`.

**参数**

- `out_chip` - Pointer to hold the initialized chip.
- `config` - Configuration of the chips to initialize.

**返回**

- `ESP_ERR_INVALID_ARG` - `out_chip` is NULL, or some field in the config is invalid.
- `ESP_ERR_NO_MEM` - failed to allocate memory for the chip structures.
- `ESP_OK` - success.
Chapter 2. API 参考

**esp_err_t spi_bus_remove_flash_device (esp_flash_t *chip)**
Remove a SPI Flash device from the SPI bus.

**参数**
- chip: The flash device to remove.

**返回**
- ESP_ERR_INVALID_ARG: The chip is invalid.
- ESP_OK: success.

**Structures**

struct esp_flash_spi_device_config_t
Configurations for the SPI Flash to init.

**Public Members**

*spi_host_device_t* **host_id**
Bus to use.

int **cs_io_num**
GPIO pin to output the CS signal.

*esp_flash_io_mode_t* **io_mode**
IO mode to read from the Flash.

enum *esp_flash_speed_s* **speed**
Speed of the Flash clock. Replaced by freq_mhz.

int **input_delay_ns**
Input delay of the data pins, in ns. Set to 0 if unknown.

int **cs_id**
CS line ID, ignored when not host_id is not SPI1_HOST, or CONFIG_SPI_FLASH_SHARE_SPI1_BUS is enabled. In this case, the CS line used is automatically assigned by the SPI bus lock.

int **freq_mhz**
The frequency of flash chip(MHZ)

**Header File**
- components/spi_flash/include/esp_flash.h

**Functions**

*esp_err_t* **esp_flash_init (esp_flash_t *chip)**
Initialise SPI flash chip interface.

This function must be called before any other API functions are called for this chip.

**备注:** Only the host and read_mode fields of the chip structure must be initialised before this function is called. Other fields may be auto-detected if left set to zero or NULL.


**Chapter 2. API 参考**

**备注:** If the chip->drv pointer is NULL, chip.chip_drv will be auto-detected based on its manufacturer & product IDs. See esp_flash_registered_flash_drivers pointer for details of this process.

参数 chip - Pointer to SPI flash chip to use. If NULL, esp.flash_default_chip is substituted. 
返回 ESP_OK on success, or a flash error code if initialisation fails.

bool esp_flash_chip_driver_initialized (const esp.flash_t *chip)  
Check if appropriate chip driver is set.

参数 chip - Pointer to SPI flash chip to use. If NULL, esp.flash_default_chip is substituted. 
返回 true if set, otherwise false.

**esp_err_t esp_flash_read_id (esp.flash_t *chip, uint32_t *out_id)**  
Read flash ID via the common “RDID” SPI flash command.

ID is a 24-bit value. Lower 16 bits of ‘id’ are the chip ID, upper 8 bits are the manufacturer ID.

参数
- chip - Pointer to identify flash chip. Must have been successfully initialised via esp.flash_init()  
- out_id - [out] Pointer to receive ID value. 
返回 ESP_OK on success, or a flash error code if operation failed.

**esp_err_t esp_flash_get_size (esp.flash_t *chip, uint32_t *out_size)**  
Detect flash size based on flash ID.

备注: 1. Most flash chips use a common format for flash ID, where the lower 4 bits specify the size as a power of 2. If the manufacturer doesn’t follow this convention, the size may be incorrectly detected.

a. The out_size returned only stands for The out_size stands for the size in the binary image header. If you want to get the real size of the chip, please call esp.flash_get_physical_size instead.

参数
- chip - Pointer to identify flash chip. Must have been successfully initialised via esp.flash_init()  
- out_size - [out] Detected size in bytes, standing for the size in the binary image header. 
返回 ESP_OK on success, or a flash error code if operation failed.

**esp_err_t esp_flash_get_physical_size (esp.flash_t *chip, uint32_t *flash_size)**  
Detect flash size based on flash ID.

备注: Most flash chips use a common format for flash ID, where the lower 4 bits specify the size as a power of 2. If the manufacturer doesn’t follow this convention, the size may be incorrectly detected.

参数
- chip - Pointer to identify flash chip. Must have been successfully initialised via esp.flash_init()  
- flash_size - [out] Detected size in bytes. 
返回 ESP_OK on success, or a flash error code if operation failed.

**esp_err_t esp_flash_read_unique_chip_id (esp.flash_t *chip, uint64_t *out_id)**  
Read flash unique ID via the common “RDUID” SPI flash command.
ID is a 64-bit value.

**参数**
- **chip** – Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init().
- **out_id** – [out] Pointer to receive unique ID value.

**返回**
- ESP_OK on success, or a flash error code if operation failed.
- ESP_ERR_NOT_SUPPORTED if the chip doesn’t support read id.

```c
esp_err_t esp_flash_erase_chip(esp_flash_t *chip)
```
Erase flash chip contents.

**参数**
- **chip** – Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()

**返回**
- ESP_OK on success,
- ESP_ERR_NOT_SUPPORTED if the chip is not able to perform the operation. This is indicated by WREN = 1 after the command is sent.
- Other flash error code if operation failed.

```c
esp_err_t esp_flash_erase_region(esp_flash_t *chip, uint32_t start, uint32_t len)
```
Erase a region of the flash chip.

Sector size is specified in chip->drv->sector_size field (typically 4096 bytes.) ESP_ERR_INVALID_ARG will be returned if the start & length are not a multiple of this size.

Erase is performed using block (multi-sector) erases where possible (block size is specified in chip->drv->block_erase_size field, typically 65536 bytes). Remaining sectors are erased using individual sector erase commands.

**参数**
- **chip** – Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init()
- **start** – Address to start erasing flash. Must be sector aligned.
- **len** – Length of region to erase. Must also be sector aligned.

**返回**
- ESP_OK on success,
- ESP_ERR_NOT_SUPPORTED if the chip is not able to perform the operation. This is indicated by WREN = 1 after the command is sent.
- Other flash error code if operation failed.

```c
esp_err_t esp_flash_get_chip_write_protect(esp_flash_t *chip, bool *write_protected)
```
Read if the entire chip is write protected.

**备注：** A correct result for this flag depends on the SPI flash chip model and chip_drv in use (via the ‘chip->drv’ field).

**参数**
- **chip** – Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init()
- **write_protected** – [out] Pointer to boolean, set to the value of the write protect flag.

**返回**
- ESP_OK on success, or a flash error code if operation failed.

```c
esp_err_t esp_flash_set_chip_write_protect(esp_flash_t *chip, bool write_protected)
```
Set write protection for the SPI flash chip.
Some SPI flash chips may require a power cycle before write protect status can be cleared. Otherwise, write protection can be removed via a follow-up call to this function.

**Note:** Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the ‘chip->drv’ field).

### Parameters
- **chip** - Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialised via esp_flash_init().
- **write_protect** - Boolean value for the write protect flag

### Return
- ESP_OK on success, or a flash error code if operation failed.

```c
esp_err_t esp_flash_get_protectable_regions(const esp_flash_t *chip, const esp_flash_region_t **out_regions, uint32_t *out_num_regions)
```

Read the list of individually protectable regions of this SPI flash chip.

**Note:** Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the ‘chip->drv’ field).

### Parameters
- **chip** - Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init().
- **out_regions** - [out] Pointer to receive a pointer to the array of protectable regions of the chip.
- **out_num_regions** - [out] Pointer to an integer receiving the count of protectable regions in the array returned in ‘regions’.

### Return
- ESP_OK on success, or a flash error code if operation failed.

```c
esp_err_t esp_flash_get_protected_region (esp_flash_t *chip, const esp_flash_region_t *region, bool *out_protected)
```

Detect if a region of the SPI flash chip is protected.

**Note:** It is possible for this result to be false and write operations to still fail, if protection is enabled for the entire chip.

### Parameters
- **chip** - Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init().
- **region** - Pointer to a struct describing a protected region. This must match one of the regions returned from esp_flash_get_protectable_regions(‘…’).
- **out_protected** - [out] Pointer to a flag which is set based on the protected status for this region.

### Return
- ESP_OK on success, or a flash error code if operation failed.

```c
esp_err_t esp_flash_set_protected_region (esp_flash_t *chip, const esp_flash_region_t *region, bool protect)
```

Update the protected status for a region of the SPI flash chip.
Chapter 2. API

**备注：** It is possible for the region protection flag to be cleared and write operations to still fail, if protection is enabled for the entire chip.

**备注：** Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the ‘chip->drv’ field).

### 参数

- **chip** – Pointer to identify flash chip. Must have been successfully initialsed via esp_flash_init()
- **region** – Pointer to a struct describing a protected region. This must match one of the regions returned from esp_flash_get_protectable_regions(
- **protect** – Write protection flag to set.

### 返回

ESP_OK on success, or a flash error code if operation failed.

**esp_err_t esp_flash_read (esp_flash_t *chip, void *buffer, uint32_t address, uint32_t length)**

Read data from the SPI flash chip.

There are no alignment constraints on buffer, address or length.

**备注：** If on-chip flash encryption is used, this function returns raw (ie encrypted) data. Use the flash cache to transparently decrypt data.

### 参数

- **chip** – Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialsed via esp_flash_init()
- **buffer** – Pointer to a buffer where the data will be read. To get better performance, this should be in the DRAM and word aligned.
- **address** – Address on flash to read from. Must be less than chip->size field.
- **length** – Length (in bytes) of data to read.

### 返回

- ESP_OK: success
- ESP_ERR_NO_MEM: Buffer is in external PSRAM which cannot be concurrently accessed, and a temporary internal buffer could not be allocated.
- or a flash error code if operation failed.

**esp_err_t esp_flash_write (esp_flash_t *chip, const void *buffer, uint32_t address, uint32_t length)**

Write data to the SPI flash chip.

There are no alignment constraints on buffer, address or length.

### 参数

- **chip** – Pointer to identify flash chip. If NULL, esp_flash_default_chip is substituted. Must have been successfully initialsed via esp_flash_init()
- **address** – Address on flash to write to. Must be previously erased (SPI NOR flash can only write bits 1->0).
- **buffer** – Pointer to a buffer with the data to write. To get better performance, this should be in the DRAM and word aligned.
- **length** – Length (in bytes) of data to write.

### 返回

- ESP_OK on success,
Chapter 2. API

- ESP_ERR_NOT_SUPPORTED if the chip is not able to perform the operation. This is indicated by WREN = 1 after the command is sent.
- Other flash error code if operation failed.

```c
esp_err_t esp_flash_write_encrypted (esp_flash_t *chip, uint32_t address, const void *buffer, uint32_t length)
```

Encrypted and write data to the SPI flash chip using on-chip hardware flash encryption.

**备注:** Both address & length must be 16 byte aligned, as this is the encryption block size

### 参数
- **chip** – Pointer to identify flash chip. Must be NULL (the main flash chip). For other chips, encrypted write is not supported.
- **address** – Address on flash to write to. 16 byte aligned. Must be previously erased (SPI NOR flash can only write bits 1->0).
- **buffer** – Pointer to a buffer with the data to write.
- **length** – Length (in bytes) of data to write. 16 byte aligned.

### 返回
- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: encrypted write not supported for this chip.
- ESP_ERR_INVALID_ARG: Either the address, buffer or length is invalid.

```c
esp_err_t esp_flash_read_encrypted (esp_flash_t *chip, uint32_t address, void *out_buffer, uint32_t length)
```

Read and decrypt data from the SPI flash chip using on-chip hardware flash encryption.

### 参数
- **chip** – Pointer to identify flash chip. Must be NULL (the main flash chip). For other chips, encrypted read is not supported.
- **address** – Address on flash to read from.
- **out_buffer** – Pointer to a buffer for the data to read to.
- **length** – Length (in bytes) of data to read.

### 返回
- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: encrypted read not supported for this chip.

```c
static inline bool esp_flash_is_quad_mode (const esp_flash_t *chip)
```

Returns true if chip is configured for Quad I/O or Quad Fast Read.

### 参数
- **chip** – Pointer to SPI flash chip to use. If NULL, esp_flash_default_chip is substituted.

### 返回
true if flash works in quad mode, otherwise false

**Structures**

```c
struct esp_flash_region_t
```

Structure for describing a region of flash.

**Public Members**

```c
uint32_t offset
```

Start address of this region.

```c
uint32_t size
```

Size of the region.
struct `esp_flash_os_functions_t`  

OS-level integration hooks for accessing flash chips inside a running OS.  

It’s in the public header because some instances should be allocated statically in the startup code. May be updated according to hardware version and new flash chip feature requirements, shouldn’t be treated as public API.  

For advanced developers, you may replace some of them with your implementations at your own risk.  

**Public Members**  

```c
esp_err_t (*start)(void *arg)
```

Called before commencing any flash operation. Does not need to be recursive (ie is called at most once for each call to ‘end’).  

```c
esp_err_t (*end)(void *arg)
```

Called after completing any flash operation.  

```c
esp_err_t (*region_protected)(void *arg, size_t start_addr, size_t size)
```

Called before any erase/write operations to check whether the region is limited by the OS.  

```c
esp_err_t (*delay_us)(void *arg, uint32_t us)
```

Delay for at least ‘us’ microseconds. Called in between ‘start’ and ‘end’.  

```c
void *(get_temp_buffer)(void *arg, size_t request_size, size_t *out_size)
```

Called for get temp buffer when buffer from application cannot be directly read into/write from.  

```c
void *(release_temp_buffer)(void *arg, void *temp_buf)
```

Called for release temp buffer.  

```c
esp_err_t (*check_yield)(void *arg, uint32_t chip_status, uint32_t *out_request)
```

Yield to other tasks. Called during erase operations.  

**Return** ESP_OK means yield needs to be called (got an event to handle), while ESP_ERR_TIMEOUT means skip yield.  

```c
esp_err_t (*yield)(void *arg, uint32_t *out_status)
```

Yield to other tasks. Called during erase operations.  

```c
int64_t (*get_system_time)(void *arg)
```

Called for get system time.  

```c
void *(set_flash_op_status)(uint32_t op_status)
```

Call to set flash operation status  

**struct `esp_flash_t`**  

Structure to describe a SPI flash chip connected to the system.  

Structure must be initialized before use (passed to esp_flash_init()). It’s in the public header because some instances should be allocated statically in the startup code. May be updated according to hardware version and new flash chip feature requirements, shouldn’t be treated as public API.  

For advanced developers, you may replace some of them with your implementations at your own risk.
Public Members

spi_flash_host_inst_t* host

Pointer to hardware-specific “host_driver” structure. Must be initialized before used.

const spi_flash_chip_t* chip_drv

Pointer to chip-model-specific “adapter” structure. If NULL, will be detected during initialisation.

const esp_flash_os_functions_t* os_func

Pointer to os-specific hook structure. Call esp_flash_init_os_functions() to setup this field, after the host is properly initialized.

void** os_func_data

Pointer to argument for os-specific hooks. Left NULL and will be initialized with os_func.

esp_flash_io_mode_t read_mode

Configured SPI flash read mode. Set before esp_flash_init is called.

uint32_t size

Size of SPI flash in bytes. If 0, size will be detected during initialisation. Note: this stands for the size in the binary image header. If you want to get the flash physical size, please call esp_flash_get_physical_size.

uint32_t chip_id

Detected chip id.

uint32_t busy

This flag is used to verify chip’s status.

uint32_t hpm_dummy_ena

This flag is used to verify whether flash works under HPM status.

uint32_t reserved_flags

reserved.

Macros

SPI_FLASH_YIELD_REQ_YIELD

SPI_FLASH_YIELD_REQ_SUSPEND

SPI_FLASH_YIELD_STA_RESUME

SPI_FLASH_OS_IS_ERASING_STATUS_FLAG

Type Definitions

typedef struct spi_flash_chip_t spi_flash_chip_t

typedef struct esp_flash_t esp_flash_t
Chapter 2. API

Header File

- components/spi_flash/include/spi_flash_mmap.h

Functions

```c
esp_err_t spi_flash_mmap (size_t src_addr, size_t size, spi_flash_mmap_memory_t memory, const void **out_ptr, spi_flash_mmap_handle_t *out_handle)
```

Map region of flash memory into data or instruction address space.

This function allocates sufficient number of 64kB MMU pages and configures them to map the requested region of flash memory into the address space. It may reuse MMU pages which already provide the required mapping.

As with any allocator, if mmap/munmap are heavily used then the address space may become fragmented. To troubleshoot issues with page allocation, use spi_flash_mmap_dump() function.

**参数**
- `src_addr` - Physical address in flash where requested region starts. This address must be aligned to 64kB boundary (SPI_FLASH_MMU_PAGE_SIZE)
- `size` - Size of region to be mapped. This size will be rounded up to a 64kB boundary
- `memory` - Address space where the region should be mapped (data or instruction)
- `out_ptr` [out] - Output, pointer to the mapped memory region
- `out_handle` [out] - Output, handle which should be used for spi_flash_munmap call

**返回**
- ESP_OK on success
- ESP_ERR_NO_MEM if pages cannot be allocated

```c
esp_err_t spi_flash_mmap_pages (const int *pages, size_t page_count, spi_flash_mmap_memory_t memory, const void **out_ptr, spi_flash_mmap_handle_t *out_handle)
```

Map sequences of pages of flash memory into data or instruction address space.

This function allocates sufficient number of 64kB MMU pages and configures them to map the indicated pages of flash memory contiguously into address space. In this respect, it works in a similar way as spi_flash_mmap() but it allows mapping a (maybe non-contiguous) set of pages into a contiguous region of memory.

**参数**
- `pages` - An array of numbers indicating the 64kB pages in flash to be mapped contiguously into memory. These indicate the indexes of the 64kB pages, not the byte-size addresses as used in other functions. Array must be located in internal memory.
- `page_count` - Number of entries in the pages array
- `memory` - Address space where the region should be mapped (instruction or data)
- `out_ptr` [out] - Output, pointer to the mapped memory region
- `out_handle` [out] - Output, handle which should be used for spi_flash_munmap call

**返回**
- ESP_OK on success
- ESP_ERR_NO_MEM if pages cannot be allocated
- ESP_ERR_INVALID_ARG if pagecount is zero or pages array is not in internal memory

```c
void spi_flash_munmap (spi_flash_mmap_handle_t handle)
```

Release region previously obtained using spi_flash_mmap.

**备注:** Calling this function will not necessarily unmap memory region. Region will only be unmapped when there are no other handles which reference this region. In case of partially overlapping regions it is possible that memory will be unmapped partially.

```c
void spi_flash_mmap_dump (void)
```

Display information about mapped regions.

This function lists handles obtained using spi_flash_mmap, along with range of pages allocated to each handle. It also lists all non-zero entries of MMU table and corresponding reference counts.
uint32_t spi_flash_mmap_get_free_pages(spi_flash_mmap_memory_t memory)

get free pages number which can be mmap

This function will return number of free pages available in mmu table. This could be useful before calling actual spi_flash_mmap (maps flash range to DCache or ICache memory) to check if there is sufficient space available for mapping.

参数 memory - memory type of MMU table free page
返回 number of free pages which can be mmaped

size_t spi_flash_cache2phys(const void* cached)

Given a memory address where flash is mapped, return the corresponding physical flash offset.

Cache address does not have have been assigned via spi_flash_mmap(), any address in memory mapped flash space can be looked up.

参数 cached - Pointer to flashed cached memory.
返回

• SPI_FLASH_CACHE2PHYS_FAIL If cache address is outside flash cache region, or the address is not mapped.
• Otherwise, returns physical offset in flash

const void *spi_flash_phys2cache(size_t phys_offs, spi_flash_mmap_memory_t memory)

Given a physical offset in flash, return the address where it is mapped in the memory space.

Physical address does not have to have been assigned via spi_flash_mmap(), any address in flash can be looked up.

备注： Only the first matching cache address is returned. If MMU flash cache table is configured so multiple entries point to the same physical address, there may be more than one cache address corresponding to that physical address. It is also possible for a single physical address to be mapped to both the IROM and DROM regions.

备注： This function doesn’t impose any alignment constraints, but if memory argument is SPI_FLASH_MMAP_INST and phys_offs is not 4-byte aligned, then reading from the returned pointer will result in a crash.

参数

• phys_offs – Physical offset in flash memory to look up.
• memory - Address space type to look up a flash cache address mapping for (instruction or data)

返回

• NULL if the physical address is invalid or not mapped to flash cache of the specified memory type.
• Cached memory address (in IROM or DROM space) corresponding to phys_offs.

Macros

ESP_ERR_FLASH_OP_FAIL

This file contains spi_flash_mmap_xx APIs, mainly for doing memory mapping to an SPI0-connected external Flash, as well as some helper functions to convert between virtual and physical address

ESP_ERR_FLASH_OP_TIMEOUT

SPI_FLASH_SEC_SIZE

SPI Flash sector size
SPI_FLASH_MMU_PAGE_SIZE
Flash cache MMU mapping page size

SPI_FLASH_CACHE2PHYS_FAIL

Type Definitions
typedef uint32_t spi_flash_mmap_handle_t
Opaque handle for memory region obtained from spi_flash_mmap.

Enumerations
enum spi_flash_mmap_memory_t
Enumeration which specifies memory space requested in an mmap call.
Values:
enumerator SPI_FLASH_MMAP_DATA
map to data memory (Vaddr0), allows byte-aligned access, 4 MB total
enumerator SPI_FLASH_MMAP_INST
map to instruction memory (Vaddr1-3), allows only 4-byte-aligned access, 11 MB total

Header File
- components/hal/include/hal/spi_flash_types.h

Structures
struct spi_flash_trans_t
Definition of a common transaction. Also holds the return value.

Public Members

uint8_t reserved
Reserved, must be 0.

uint8_t mosi_len
Output data length, in bytes.

uint8_t miso_len
Input data length, in bytes.

uint8_t address_bitlen
Length of address in bits, set to 0 if command does not need an address.

uint32_t address
Address to perform operation on.
Chapter 2. API 参考

const uint8_t *mosi_data
  Output data to salve.

uint8_t *miso_data
  [out] Input data from slave, little endian

uint32_t flags
  Flags for this transaction. Set to 0 for now.

uint16_t command
  Command to send.

uint8_t dummy_bitlen
  Basic dummy bits to use.

uint32_t io_mode
  Flash working mode when SPI_FLASH_IGNORE_BASEIO is specified.

struct spi_flash_sus_cmd_conf
  Configuration structure for the flash chip suspend feature.
  
  **Public Members**

  uint32_t sus_mask
    SUS/SUS1/SUS2 bit in flash register.

  uint32_t cmd_rdsr
    Read flash status register(2) command.

  uint32_t sus_cmd
    Flash suspend command.

  uint32_t res_cmd
    Flash resume command.

  uint32_t reserved
    Reserved, set to 0.

struct spi_flash_encryption_t
  Structure for flash encryption operations.
  
  **Public Members**

  void (*flash_encryption_enable)(void)
    Enable the flash encryption.

  void (*flash_encryption_disable)(void)
    Disable the flash encryption.
Chapter 2. API

void (*flash_encryption_data_prepare)(uint32_t address, const uint32_t *buffer, uint32_t size)
Prepare flash encryption before operation.

备注: address and buffer must be 8-word aligned.

Param address  The destination address in flash for the write operation.
Param buffer   Data for programming
Param size     Size to program.

void (*flash_encryption_done)(void)
flash data encryption operation is done.

void (*flash_encryption_destroy)(void)
Destroy encrypted result

bool (*flash_encryption_check)(uint32_t address, uint32_t length)
Check if is qualified to encrypt the buffer

Param address  the address of written flash partition.
Param length   Buffer size.

struct spi_flash_host_inst_t
SPI Flash Host driver instance

Public Members

const struct spi_flash_host_driver_s *driver
Pointer to the implementation function table.

struct spi_flash_host_driver_s
Host driver configuration and context structure.

Public Members

esp_err_t (*dev_config)(spi_flash_host_inst_t *host)
Configure the device-related register before transactions. This saves some time to re-configure those registers when we send continuously

esp_err_t (*common_command)(spi_flash_host_inst_t *host, spi_flash_trans_t *t)
Send an user-defined spi transaction to the device.

esp_err_t (*read_id)(spi_flash_host_inst_t *host, uint32_t *id)
Read flash ID.

void (*erase_chip)(spi_flash_host_inst_t *host)
Erase whole flash chip.
void (*erase_sector)(spi_flash_host_inst_t *host, uint32_t start_address)
   Erase a specific sector by its start address.

void (*erase_block)(spi_flash_host_inst_t *host, uint32_t start_address)
   Erase a specific block by its start address.

esp_err_t (*read_status)(spi_flash_host_inst_t *host, uint8_t *out_sr)
   Read the status of the flash chip.

esp_err_t (*set_write_protect)(spi_flash_host_inst_t *host, bool wp)
   Disable write protection.

void (*program_page)(spi_flash_host_inst_t *host, const void *buffer, uint32_t address, uint32_t length)
   Program a page of the flash. Check max_write_bytes for the maximum allowed writing length.

bool (*supports_direct_write)(spi_flash_host_inst_t *host, const void *p)
   Check whether the SPI host supports direct write.
   When cache is disabled, SPI1 doesn’t support directly write when buffer isn’t internal.

int (*write_data_slicer)(spi_flash_host_inst_t *host, uint32_t address, uint32_t len, uint32_t *align_addr, uint32_t page_size)
   Slicer for write data. The program_page should be called iteratively with the return value of this function.

   Param address  Beginning flash address to write
   Param len     Length request to write
   Param align_addr Output of the aligned address to write to
   Param page_size Physical page size of the flash chip
   Return Length that can be actually written in one program_page call

esp_err_t (*read)(spi_flash_host_inst_t *host, void *buffer, uint32_t address, uint32_t read_len)
   Read data from the flash. Check max_read_bytes for the maximum allowed reading length.

bool (*supports_direct_read)(spi_flash_host_inst_t *host, const void *p)
   Check whether the SPI host supports direct read.
   When cache is disabled, SPI1 doesn’t support directly read when the given buffer isn’t internal.

int (*read_data_slicer)(spi_flash_host_inst_t *host, uint32_t address, uint32_t len, uint32_t *align_addr, uint32_t page_size)
   Slicer for read data. The read should be called iteratively with the return value of this function.

   Param address  Beginning flash address to read
   Param len     Length request to read
   Param align_addr Output of the aligned address to read
   Param page_size Physical page size of the flash chip
   Return Length that can be actually read in one read call

uint32_t (*host_status)(spi_flash_host_inst_t *host)
   Check the host status, 0:busy, 1:idle, 2:suspended.

esp_err_t (*configure_host_io_mode)(spi_flash_host_inst_t *host, uint32_t command, uint32_t addr_bitlen, int dummy_bitlen_base, esp_flash_io_mode_t io_mode)
Configure the host to work at different read mode. Responsible to compensate the timing and set IO mode.

```c
void (*poll_cmd_done)(spi_flash_host_inst_t *host)
```

Internal use, poll the HW until the last operation is done.

```c
esp_err_t (*flush_cache)(spi_flash_host_inst_t *host, uint32_t addr, uint32_t size)
```

For some host (SPI1), they are shared with a cache. When the data is modified, the cache needs to be flushed. Left NULL if not supported.

```c
void (*check_suspend)(spi_flash_host_inst_t *host)
```

Suspend check erase/program operation, reserved for ESP32-C3 and ESP32-S3 spi flash ROM IMPL.

```c
void (*resume)(spi_flash_host_inst_t *host)
```

Resume flash from suspend manually

```c
void (*suspend)(spi_flash_host_inst_t *host)
```

Set flash in suspend status manually

```c
esp_err_t (*sus_setup)(spi_flash_host_inst_t *host, const spi_flash_sus_cmd_conf *sus_conf)
```

Suspend feature setup for setting cmd and status register mask.

Macros

```c
#define SPI_FLASH_TRANS_FLAG_CMD16
```

Send command of 16 bits.

```c
#define SPI_FLASH_TRANS_FLAG_IGNORE_BASEIO
```

Not applying the basic io mode configuration for this transaction.

```c
#define SPI_FLASH_TRANS_FLAG_BYTE_SWAP
```

Used for DTR mode, to swap the bytes of a pair of rising/falling edge.

```c
#define SPI_FLASH_CONFIG_CONF_BITS
```

OR the io_mode with this mask, to enable the dummy output feature or replace the first several dummy bits into address to meet the requirements of conf bits. (Used in DIO/QIO/OIO mode)

```c
#define SPI_FLASH_OPI_FLAG
```

A flag for flash work in opi mode, the io mode below are opi, above are SPI/QSPI mode. DO NOT use this value in any API.

```c
#define SPI_FLASH_READ_MODE_MIN
```

Slowest io mode supported by ESP32, currently SlowRd.

Type Definitions

typedef enum esp_flash_speed_s esp_flash_speed_t

SPI flash clock speed values, always refer to them by the enum rather than the actual value (more speed may be appended into the list).

A strategy to select the maximum allowed speed is to enumerate from the ESP_FLASH_SPEED_MAX-1 or highest frequency supported by your flash, and decrease the speed until the probing success.
typedef struct spi_flash_host_driver_s spi_flash_host_driver_t

Enumerations

enum esp_flash_speed_s
SPI flash clock speed values, always refer to them by the enum rather than the actual value (more speed may be appended into the list).

A strategy to select the maximum allowed speed is to enumerate from the ESP_FLASH_SPEED_MAX-1 or highest frequency supported by your flash, and decrease the speed until the probing success.

Values:

enumerator ESP_FLASH_5MHZ
The flash runs under 5MHz.

enumerator ESP_FLASH_10MHZ
The flash runs under 10MHz.

enumerator ESP_FLASH_20MHZ
The flash runs under 20MHz.

enumerator ESP_FLASH_26MHZ
The flash runs under 26MHz.

enumerator ESP_FLASH_40MHZ
The flash runs under 40MHz.

enumerator ESP FLASH _80MHZ
The flash runs under 80MHz.

enumerator ESP FLASH _120MHZ
The flash runs under 120MHz, 120MHZ can only be used by main flash after timing tuning in system. Do not use this directly in any API.

enumerator ESP FLASH _SPEED_MAX
The maximum frequency supported by the host is ESP_FLASH_SPEED_MAX-1.

enum esp_flash_io_mode_t
Mode used for reading from SPI flash.

Values:

enumerator SPI_FLASH_SLOWRD
Data read using single I/O, some limits on speed.

enumerator SPI_FLASH_FASTRD
Data read using single I/O, no limit on speed.

enumerator SPI_FLASH_DOUT
Data read using dual I/O.
enumerator **SPI_FLASH_DIO**
Both address & data transferred using dual I/O.

enumerator **SPI_FLASH_QOUT**
Data read using quad I/O.

enumerator **SPI_FLASH_QIO**
Both address & data transferred using quad I/O.

enumerator **SPI_FLASH_OPI_STR**
Only support on OPI flash, flash read and write under STR mode.

enumerator **SPI_FLASH_OPI_DTR**
Only support on OPI flash, flash read and write under DTR mode.

enumerator **SPI_FLASH_READ_MODE_MAX**
The fastest io mode supported by the host is `ESP_FLASH_READ_MODE_MAX-1`.

**Header File**

- components/hal/include/hal/esp_flash_err.h

**Macros**

**ESP_ERR_FLASH_NOTInicialised**
`esp_flash_chip_t` structure not correctly initialised by `esp_flash_init()`.

**ESP_ERR_FLASH_UNSUPPORTED_HOST**
Requested operation isn’t supported via this host SPI bus (chip->spi field).

**ESP_ERR_FLASH_UNSUPPORTED_CHIP**
Requested operation isn’t supported by this model of SPI flash chip.

**ESP_ERR_FLASH_PROTECTED**
Write operation failed due to chip’s write protection being enabled.

**Enumerations**

enum [anonymous]

Values:

enumerator **ESP_ERR_FLASH_SIZE_NOT_MATCH**
The chip doesn’t have enough space for the current partition table.

enumerator **ESP_ERR_FLASH_NO_RESPONSE**
Chip did not respond to the command, or timed out.
Chapter 2. API 参考

Header File

- components/esp_partition/include/esp_partition.h

Functions

- esp_partition_iterator_t esp_partition_find(esp_partition_type_t type, esp_partition_subtype_t subtype, const char *label)

  Find partition based on one or more parameters.

  参数
  - type - Partition type, one of esp_partition_type_t values or an 8-bit unsigned integer. To find all partitions, no matter the type, use ESP_PARTITION_TYPE_ANY, and set subtype argument to ESP_PARTITION_SUBTYPE_ANY.
  - subtype - Partition subtype, one of esp_partition_subtype_t values or an 8-bit unsigned integer. To find all partitions of given type, use ESP_PARTITION_SUBTYPE_ANY.
  - label - (optional) Partition label. Set this value if looking for partition with a specific name. Pass NULL otherwise.

  返回 iterator which can be used to enumerate all the partitions found, or NULL if no partitions were found. Iterator obtained through this function has to be released using esp_partition_iterator_release when not used any more.

- const esp_partition_t *esp_partition_find_first(esp_partition_type_t type, esp_partition_subtype_t subtype, const char *label)

  Find first partition based on one or more parameters.

  参数
  - type - Partition type, one of esp_partition_type_t values or an 8-bit unsigned integer. To find all partitions, no matter the type, use ESP_PARTITION_TYPE_ANY, and set subtype argument to ESP_PARTITION_SUBTYPE_ANY.
  - subtype - Partition subtype, one of esp_partition_subtype_t values or an 8-bit unsigned integer. To find all partitions of given type, use ESP_PARTITION_SUBTYPE_ANY.
  - label - (optional) Partition label. Set this value if looking for partition with a specific name. Pass NULL otherwise.

  返回 pointer to esp_partition_t structure, or NULL if no partition is found. This pointer is valid for the lifetime of the application.

- const esp_partition_t *esp_partition_get(esp_partition_iterator_t iterator)

  Get esp_partition_t structure for given partition.

  参数 iterator - Iterator obtained using esp_partition_find. Must be non-NULL.

  返回 pointer to esp_partition_t structure. This pointer is valid for the lifetime of the application.

- esp_partition_iterator_t esp_partition_next(esp_partition_iterator_t iterator)

  Move partition iterator to the next partition found.

  Any copies of the iterator will be invalid after this call.

  参数 iterator - Iterator obtained using esp_partition_find. Must be non-NULL.

  返回 NULL if no partition was found, valid esp_partition_iterator_t otherwise.

- void esp_partition_iterator_release(esp_partition_iterator_t iterator)

  Release partition iterator.

  参数 iterator - Iterator obtained using esp_partition_find. The iterator is allowed to be NULL, so it is not necessary to check its value before calling this function.

- const esp_partition_t *esp_partition_verify(const esp_partition_t *partition)

  Verify partition data.

  Given a pointer to partition data, verify this partition exists in the partition table (all fields match.)
This function is also useful to take partition data which may be in a RAM buffer and convert it to a pointer to the permanent partition data stored in flash.

Pointers returned from this function can be compared directly to the address of any pointer returned from `esp_partition_get()`, as a test for equality.

参数 partition – Pointer to partition data to verify. Must be non-NULL. All fields of this structure must match the partition table entry in flash for this function to return a successful match.

返回
  • If partition not found, returns NULL.
  • If found, returns a pointer to the `esp_partition_t` structure in flash. This pointer is always valid for the lifetime of the application.

`esp_err_t esp_partition_read` (const `esp_partition_t *partition`, size_t src_offset, void *dst, size_t size)
Read data from the partition.

Partitions marked with an encryption flag will automatically be be read and decrypted via a cache mapping.

参数
  • partition – Pointer to partition structure obtained using `esp_partition_find_first` or `esp_partition_get`. Must be non-NULL.
  • dst – Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least ‘size’ bytes long.
  • src_offset – Address of the data to be read, relative to the beginning of the partition.
  • size – Size of data to be read, in bytes.

返回 ESP_OK, if data was read successfully; ESP_ERR_INVALID_ARG, if src_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition; or one of error codes from lower-level flash driver.

`esp_err_t esp_partition_write` (const `esp_partition_t *partition`, size_t dst_offset, const void *src, size_t size)
Write data to the partition.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using `esp_partition_erase_range` function.

Partitions marked with an encryption flag will automatically be written via the `esp_flash_write_encrypted()` function. If writing to an encrypted partition, all write offsets and lengths must be multiples of 16 bytes. See the `esp_flash_write_encrypted()` function for more details. Unencrypted partitions do not have this restriction.

备注: Prior to writing to flash memory, make sure it has been erased with `esp_partition_erase_range` call.

参数
  • partition – Pointer to partition structure obtained using `esp_partition_find_first` or `esp_partition_get`. Must be non-NULL.
  • dst_offset – Address where the data should be written, relative to the beginning of the partition.
  • src – Pointer to the source buffer. Pointer must be non-NULL and buffer must be at least ‘size’ bytes long.
  • size – Size of data to be written, in bytes.

返回 ESP_OK, if data was written successfully; ESP_ERR_INVALID_ARG, if dst_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition; or one of error codes from lower-level flash driver.

`esp_err_t esp_partition_read_raw` (const `esp_partition_t *partition`, size_t src_offset, void *dst, size_t size)
Read data from the partition without any transformation/decryption.
This function is essentially the same as `esp_partition_read()` above. It just never decrypts data but returns it as is.

**Parameters**
- `partition` – Pointer to partition structure obtained using `esp_partition_find_first` or `esp_partition_get`. Must be non-NULL.
- `dst` – Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least `size` bytes long.
- `src_offset` – Address of the data to be read, relative to the beginning of the partition.
- `size` – Size of data to be read, in bytes.

**Return**
- ESP_OK, if data was read successfully; ESP_ERR_INVALID_ARG, if `src_offset` exceeds partition size; ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition; or one of error codes from lower-level flash driver.

```c
esp_err_t esp_partition_write_raw(const esp_partition_t *partition, size_t dst_offset, const void *src, size_t size)
```

Write data to the partition without any transformation/encryption.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using `esp_partition_erase_range` function.

This function is essentially the same as `esp_partition_write()` above. It just never encrypts data but writes it as is.

**Parameters**
- `partition` – Pointer to partition structure obtained using `esp_partition_find_first` or `esp_partition_get`. Must be non-NULL.
- `dst_offset` – Address where the data should be written, relative to the beginning of the partition.
- `src` – Pointer to the source buffer. Pointer must be non-NULL and buffer must be at least `size` bytes long.
- `size` – Size of data to be written, in bytes.

**Return**
- ESP_OK, if data was written successfully; ESP_ERR_INVALID_ARG, if `dst_offset` exceeds partition size; ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition; or one of error codes from lower-level flash driver.

```c
esp_err_t esp_partition_erase_range(const esp_partition_t *partition, size_t offset, size_t size)
```

Erase part of the partition.

Prior to writing to flash memory, make sure it has been erased with `esp_partition_erase_range` call.

This function is essentially the same as `esp_partition_write()` above. It just never encrypts data but writes it as is.

**Parameters**
- `partition` – Pointer to partition structure obtained using `esp_partition_find_first` or `esp_partition_get`. Must be non-NULL.
- `offset` – Offset from the beginning of partition where erase operation should start. Must be aligned to `partition->erase_size`.
- `size` – Size of the range which should be erased, in bytes. Must be divisible by `partition->erase_size`.

**Return**
- ESP_OK, if the range was erased successfully; ESP_ERR_INVALID_ARG, if iterator or `dst` are NULL; ESP_ERR_INVALID_SIZE, if `erase` would go out of bounds of the partition; or one of error codes from lower-level flash driver.
**esp_err_t esp_partition_mmap**

(const esp_partition_t *partition, size_t offset, size_t size,
 esp_partition_mmap_memory_t memory, const void **out_ptr,
 esp_partition_mmap_handle_t *out_handle)

Configure MMU to map partition into data memory.

Unlike spi_flash_mmap function, which requires a 64kB aligned base address, this function doesn’t impose such a requirement. If offset results in a flash address which is not aligned to 64kB boundary, address will be rounded to the lower 64kB boundary, so that mapped region includes requested range. Pointer returned via out_ptr argument will be adjusted to point to the requested offset (not necessarily to the beginning of mmap-ed region).

To release mapped memory, pass handle returned via out_handle argument to esp_partition_munmap function.

**参数**
- `partition` – Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- `offset` – Offset from the beginning of partition where mapping should start.
- `size` – Size of the area to be mapped.
- `memory` – Memory space where the region should be mapped
- `out_ptr` – Output, pointer to the mapped memory region
- `out_handle` – Output, handle which should be used for esp_partition_munmap call

**返回** ESP_OK, if successful

void **esp_partition_munmap** 

(esp_partition_mmap_handle_t handle)

Release region previously obtained using esp_partition_mmap.

---

**备注：** Calling this function will not necessarily unmap memory region. Region will only be unmapped when there are no other handles which reference this region. In case of partially overlapping regions it is possible that memory will be unmapped partially.

---

**参数** handle – Handle obtained from spi_flash_mmap

**esp_err_t esp_partition_get_sha256**

(const esp_partition_t *partition, uint8_t *sha_256)

Get SHA-256 digest for required partition.

For apps with SHA-256 appended to the app image, the result is the appended SHA-256 value for the app image content. The hash is verified before returning, if app content is invalid then the function returns ESP_ERR_IMAGE_INVALID. For apps without SHA-256 appended to the image, the result is the SHA-256 of all bytes in the app image. For other partition types, the result is the SHA-256 of the entire partition.

**参数**
- `partition` – [in] Pointer to info for partition containing app or data. (fields: address, size and type, are required to be filled).
- `sha_256` – [out] Returned SHA-256 digest for a given partition.

**返回**
- ESP_OK: In case of successful operation.
- ESP_ERR_IMAGE_INVALID_ARG: The size was 0 or the sha_256 was NULL.
- ESP_ERR_NO_MEM: Cannot allocate memory for sha256 operation.
- ESP_ERR_IMAGE_INVALID: App partition doesn’t contain a valid app image.
- ESP_FAIL: An allocation error occurred.

**bool esp_partition_check_identity**

(const esp_partition_t *partition_1, const esp_partition_t *partition_2)

Check for the identity of two partitions by SHA-256 digest.

**参数**
- `partition_1` – [in] Pointer to info for partition 1 containing app or data. (fields: address, size and type, are required to be filled).
- `partition_2` – [in] Pointer to info for partition 2 containing app or data. (fields: address, size and type, are required to be filled).
Register a partition on an external flash chip.

This API allows designating certain areas of external flash chips (identified by the `esp_flash_t` structure) as partitions. This allows using them with components which access SPI flash through the esp_partition API.

### Parameters

- **flash_chip** - Pointer to the structure identifying the flash chip
- **offset** - Address in bytes, where the partition starts
- **size** - Size of the partition in bytes
- **label** - Partition name
- **type** - One of the partition types (ESP_PARTITION_TYPE_*), or an integer. Note that applications cannot be booted from external flash chips, so using ESP_PARTITION_TYPE_APP is not supported.
- **subtype** - One of the partition subtypes (ESP_PARTITION_SUBTYPE_*), or an integer.
- **out_partition** - [out] Output, if non-NULL, receives the pointer to the resulting `esp_partition_t` structure

### Return

- ESP_OK on success
- ESP_ERR_NO_MEM if memory allocation has failed
- ESP_ERR_INVALID_ARG if the new partition overlaps another partition on the same flash chip
- ESP_ERR_INVALID_SIZE if the partition doesn’t fit into the flash chip size

Deregister the partition previously registered using the `esp_partition_register_external` function.

### Structures

**struct esp_partition_t**

Partition information structure

This is not the format in flash, that format is esp_partition_info_t.

However, this is the format used by this API.

### Public Members

**esp_flash_t *flash_chip**

SPI flash chip on which the partition resides
**Chapter 2. API**

*esp_partition_type_t* `type`
partition type (app/data)

*esp_partition_subtype_t* `subtype`
partition subtype

`uint32_t` `address`
starting address of the partition in flash

`uint32_t` `size`
size of the partition, in bytes

`uint32_t` `erase_size`
size the erase operation should be aligned to

`char` `label[17]`
partition label, zero-terminated ASCII string

`bool` `encrypted`
flag is set to true if partition is encrypted

**Macros**

`ESP_PARTITION_SUBTYPE_OTA(i)`
Convenience macro to get `esp_partition_subtype_t` value for the i-th OTA partition.

**Type Definitions**

typedef `uint32_t` `esp_partition_mmap_handle_t`
Opaque handle for memory region obtained from `esp_partition_mmap`.

typedef `struct esp_partition_iterator_opaque_*` `esp_partition_iterator_t`
Opaque partition iterator type.

**Enumerations**

typedef enum `esp_partition_mmap_memory_t`
Enumeration which specifies memory space requested in an mmap call.

*Values:*

- enumerator `ESP_PARTITION_MMAP_DATA`
  map to data memory (Vaddr0), allows byte-aligned access, 4 MB total

- enumerator `ESP_PARTITION_MMAP_INST`
  map to instruction memory (Vaddr1-3), allows only 4-byte-aligned access, 11 MB total

typedef enum `esp_partition_type_t`
Partition type.
Partition types with integer value 0x00-0x3F are reserved for partition types defined by ESP-IDF. Any other integer value 0x40-0xFE can be used by individual applications, without restriction.

Values:

enumerator `ESP_PARTITION_TYPE_APP`
Application partition type.

data type `ESP_PARTITION_TYPE_DATA`
Data partition type.

data type `ESP_PARTITION_TYPE_ANY`
Used to search for partitions with any type.

enum `esp_partition_subtype_t`
Partition subtype.

Application-defined partition types (0x40-0xFE) can set any numeric subtype value.

These ESP-IDF-defined partition subtypes apply to partitions of type ESP_PARTITION_TYPE_APP and ESP_PARTITION_TYPE_DATA.

Values:

enumerator `ESP_PARTITION_SUBTYPE_APP_FACTORY`
Factory application partition.

data type `ESP_PARTITION_SUBTYPE_APP_OTA_MIN`
Base for OTA partition subtypes.

data type `ESP_PARTITION_SUBTYPE_APP_OTA_0`
OTA partition 0.

data type `ESP_PARTITION_SUBTYPE_APP_OTA_1`
OTA partition 1.

data type `ESP_PARTITION_SUBTYPE_APP_OTA_2`
OTA partition 2.

data type `ESP_PARTITION_SUBTYPE_APP_OTA_3`
OTA partition 3.

data type `ESP_PARTITION_SUBTYPE_APP_OTA_4`
OTA partition 4.

data type `ESP_PARTITION_SUBTYPE_APP_OTA_5`
OTA partition 5.
enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_6**
OTA partition 6.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_7**
OTA partition 7.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_8**
OTA partition 8.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_9**
OTA partition 9.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_10**
OTA partition 10.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_11**
OTA partition 11.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_12**
OTA partition 12.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_13**
OTA partition 13.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_14**
OTA partition 14.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_15**
OTA partition 15.

enumerator **ESP_PARTITION_SUBTYPE_APP_OTA_MAX**
Max subtype of OTA partition.

enumerator **ESP_PARTITION_SUBTYPE_APP_TEST**
Test application partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_OTA**
OTA selection partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_PHY**
PHY init data partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_NVS**
NVS partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_COREDUMP**
COREDUMP partition.
enumerator **ESP_PARTITION_SUBTYPE_DATA_NVS_KEYS**
Partition for NVS keys.

enumerator **ESP_PARTITION_SUBTYPE_DATA_EFUSE_EM**
Partition for emulate eFuse bits.

enumerator **ESP_PARTITION_SUBTYPE_DATA_UNDEFINED**
Undefined (or unspecified) data partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_EFUSE_EFUSE**
ESP_FUSE partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_FAT**
FAT partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_SPIFFS**
SPIFFS partition.

enumerator **ESP_PARTITION_SUBTYPE_DATA_ANY**
Used to search for partitions with any subtype.

**Flash 加密 API 参考**

**Header File**

- components/bootloader_support/include/esp_flash_encrypt.h

**Functions**

bool **esp_flash_encryption_enabled** (void)
Is flash encryption currently enabled in hardware?
Flash encryption is enabled if the FLASH_CRYPT_CNT efuse has an odd number of bits set.

- 返回 true if flash encryption is enabled.

```c
esp_err_t esp_flash_encrypt_check_and_update (void)
```

bool **esp_flash_encrypt_state** (void)
Returns the Flash Encryption state and prints it.

- 返回 True - Flash Encryption is enabled False - Flash Encryption is not enabled

bool **esp_flash_encrypt_initialized_once** (void)
Checks if the first initialization was done.
If the first initialization was done then FLASH_CRYPT_CNT != 0

- 返回 true - the first initialization was done false - the first initialization was NOT done

```c
esp_err_t esp_flash_encrypt_init (void)
The first initialization of Flash Encryption key and related eFuses.
```

- 返回 ESP_OK if all operations succeeded

```c
esp_err_t esp_flash_encrypt_contents (void)
Encrypts flash content.
```

- 返回 ESP_OK if all operations succeeded
**esp_err_t** **esp_flash_encrypt_enable** (void)

Activates Flash encryption on the chip.

It burns FLASH_CRYPT_CNT eFuse based on the CONFIG_SECURE_FLASH_ENCRYPTION_MODE_RELEASE option.

返回 ESP_OK if all operations succeeded

**bool** **esp_flash_encrypt_is_write_protected** (bool print_error)

Returns True if the write protection of FLASH_CRYPT_CNT is set.

参数 **print_error** - Print error if it is write protected

返回 true - if FLASH_CRYPT_CNT is write protected

**esp_err_t** **esp_flash_encrypt_region** (uint32_t src_addr, size_t data_length)

Encrypt-in-place a block of flash sectors.

备注: This function resets RTC_WDT between operations with sectors.

参数

- **src_addr** - Source offset in flash. Should be multiple of 4096 bytes.
- **data_length** - Length of data to encrypt in bytes. Will be rounded up to next multiple of 4096 bytes.

返回 ESP_OK if all operations succeeded, ESP_ERR_FLASH_OP_FAIL if SPI flash fails, ESP_ERR_FLASH_OP_TIMEOUT if flash times out.

**void** **esp_flash_write_protect_crypt_cnt** (void)

Write protect FLASH_CRYPT_CNT.

Intended to be called as a part of boot process if flash encryption is enabled but secure boot is not used. This should protect against serial re-flashing of an unauthorised code in absence of secure boot.

备注: On ESP32 V3 only, write protecting FLASH_CRYPT_CNT will also prevent disabling UART Download Mode. If both are wanted, call esp_efuse_disable_rom_download_mode() before calling this function.

**esp_flash_enc_mode_t** **esp_get_flash_encryption_mode** (void)

Return the flash encryption mode.

The API is called during boot process but can also be called by application to check the current flash encryption mode of ESP32

返回

**void** **esp_flash_encryption_init_checks** (void)

Check the flash encryption mode during startup.

Verifies the flash encryption config during startup:

- Correct any insecure flash encryption settings if hardware Secure Boot is enabled.
- Log warnings if the efuse config doesn’t match the project config in any way

备注: This function is called automatically during app startup, it doesn’t need to be called from the app.
```c
esp_err_t esp_flash_encryption_enable_secure_features (void)
Set all secure eFuse features related to flash encryption.

- ESP_OK - Successfully

void esp_flash_encryption_set_release_mode (void)
Switches Flash Encryption from “Development” to “Release”.
If already in “Release” mode, the function will do nothing. If flash encryption efuse is not enabled yet then abort. It burns:
- ” disable encrypt in dl mode”
- set FLASH_CRYPT_CNT efuse to max

Enumerations
enum esp_flash_enc_mode_t
Values:
- enumerator ESP_FLASH_ENC_MODE_DISABLED
- enumerator ESP_FLASH_ENC_MODE_DEVELOPMENT
- enumerator ESP_FLASH_ENC_MODE_RELEASE

2.9.8 SPIFFS 文件系统

概述
SPIFFS 是一个用于 SPI NOR flash 设备的嵌入式文件系统，支持磨损均衡、文件系统一致性检查等功能。

说明
- 目前，SPIFFS 尚不支持目录，但可以生成扁平结构。如果 SPIFFS 挂载在 /spiffs 下，在 /spiffs/tmp/myfile.txt 路径下创建一个文件则会在 SPIFFS 中生成一个名为 /tmp/myfile.txt 的文件，而不是在 /spiffs/tmp下生成名为 myfile.txt 的文件
- SPIFFS 并非实时找，每次写操作耗时不等
- 目前，SPIFFS 尚不支持检测或处理已损坏的块。
- SPIFFS 只能稳定地使用约 75% 的指定分区容量。
- 当文件系统空间不足时，垃圾收集器会尝试多次扫描文件系统来寻找可用空间。根据所需空间的不同，写操作会被调用多次，每次函数调用将花费几秒。同一操作可能会花费不同长的时间缘于 SPIFFS 的设计，且已在官方的 SPIFFS github 仓库或是 <https://github.com/espressif/esp-idf/issues/1737> 中被多次报告。这个问题可以通过 SPIFFS 配置部分缓解。
- 被删除文件通常不会被完全清除，会在文件系统中遗留下无法使用的部分。
- 如果 ESP32 在文件系统操作期间断电，可能会导致 SPIFFS 损坏。但通过 esp_spiffs_check 函数恢复文件系统。详情请参阅官方 SPIFFS FAQ <https://github.com/pellepl/spiffs/wiki/FAQ>。
## 工具

**spiffsgen.py**

```python
python spiffsgen.py <image_size> <base_dir> <output_file>
```

参数（必选）说明如下：
- **image_size**: 分区大小，用于烧录生成的 SPIFFS 镜像；
- **base_dir**: 创建 SPIFFS 镜像的目录；
- **output_file**: SPIFFS 镜像输出文件。

其他参数（可选）也参与控制镜像的生成，用户可以运行以下帮助命令，查看这些参数的具体信息：

```python
python spiffsgen.py --help
```

上述可选参数对应 SPIFFS 构建配置选项。若想顺利生成可用的镜像，请确保使用的参数或配置与构建 SPIFFS 时所用的参数或配置相同。运行帮助命令将显示参数所对应的 SPIFFS 构建配置。如未指定参数，将使用帮助信息中的默认值。

镜像生成后，用户可以使用 esptool.py 或 parttool.py 烧录镜像。

用户可以在命令行或脚本中手动单独调用 spiffsgen.py，也可以直接从构建系统调用 spiffs_create_partition_image 来使用 spiffsgen.py:

```python
spiffs_create_partition_image(<partition> <base_dir> [FLASH_IN_PROJECT] [DEPENDS_.—dep dep dep...])
```

在构建系统中使用 spiffsgen.py 更为方便，构建配置会自动传递给 spiffsgen.py 工具，确保生成的镜像可用于构建。比如，单独调用 spiffsgen.py 时需要用到 **image_size** 参数，但在构建系统中调用 spiffs_create_partition_image 时，仅需要 **partition** 参数，镜像大小将直接从工程分区表中获取。

使用 spiffs_create_partition_image，必须从组件 CMakeLists.txt 文件调用。

用户也可以指定 FLASH_IN_PROJECT，然后使用 idf.py flash 将镜像与应用程序二进制文件、分区表等一起自动烧录至设备，例如：

```python
spiffs_create_partition_image(my_spiffs_partition my_folder FLASH_IN_PROJECT)
```

指定 FLASH_IN_PROJECT/SPIFFS_IMAGE_FLASH_IN_PROJECT 也可以生成镜像，但须使用 esptool.py、parttool.py 或自定义构建系统目标手动烧录。

有时基本目录中的内容是在构建时生成的，用户可以使用 DEPENDS/SPIFFS_IMAGE_DEPENDS 指定目标，因此可以在生成镜像之前执行此目标：

```python
add_custom_target(dep COMMAND ...)
spiffs_create_partition_image(my_spiffs_partition my_folder DEPENDS dep)
```

请参考 storage/spiffsgen，查看示例。

**mkspiffs** 用户也可以使用 mkspiffs 工具创建 SPIFFS 分区镜像。与 spiffsgen.py 相似，mkspiffs 也可以用于从指定文件夹中生成镜像，然后使用 esptool.py 烧录镜像。

该工具需要获取以下参数：
- **Block Size**: 4096（SPI flash 标准）
- **Page Size**: 256（SPI flash 标准）
- **Image Size**: 分区大小（以字节为单位，可从分区表中获取）
- **Partition Offset**: 分区起始地址（可从分区表中获取）

运行以下命令，将文件夹打包成 1 MB 大小的镜像:

```bash
mkspiffs -i image_size -o base_dir -o output_file
```
选择合适的 SPIFFS 工具  上面介绍的两款 SPIFFS 工具功能相似，需根据实际情况，选择合适的一款。 以下情况优先选用 spiffsgen.py 工具：

1. 仅需在构建时简单生成 SPIFFS 镜像，请选择使用 spiffsgen.py，因为 spiffsgen.py 可以直接在构建系统中使用函数或命令生成 SPIFFS 镜像。
2. 主机没有可用的 C/C++ 编译器时，可以选择使用 spiffsgen.py 工具，因为 spiffsgen.py 不需要编译。

以下情况优先选用 mkspiffs 工具：

1. 如果用户除了需要生成镜像外，还需要拆包 SPIFFS 镜像，请选择使用 mkspiffs 工具，因为 spiffsgen.py 目前尚不支持此功能。
2. 如果用户当前环境中 Python 解释器不可用，但主机编译器可用，或者有预编译的 mkspiffs 二进制文件，此时请选择使用 mkspiffs 工具。但是，mkspiffs 没有集成到构建系统，用户必须自己完成以下工作：在构建期间编译 mkspiffs（如果未使用预编译的二进制文件），为输出文件创建构建规则或目标，将适当的参数传递给工具等。

另请参阅

- 分区表

应用示例

storage/spiffs 目录下提供了 SPIFFS 应用示例。该示例初始化并挂载了一个 SPIFFS 分区，然后使用 POSIX 和 C 库写入和读取数据。请参考 example 目录下的 README.md 文件，获取详细信息。

高级 API 参考

Header File

- components/spiffs/include/esp_spiffs.h

Functions

```c
esp_err_t esp_vfs_spiffs_register(const esp_vfs_spiffs_conf_t *conf)
```

Register and mount SPIFFS to VFS with given path prefix.

- 参数 conf – Pointer to esp_vfs_spiffs_conf_t configuration structure
- 返回 ESP_OK if success
- ESP_ERR_NO_MEM if objects could not be allocated
- ESP_ERR_INVALID_STATE if already mounted or partition is encrypted
- ESP_ERR_NOT_FOUND if partition for SPIFFS was not found
- ESP_FAIL if mount or format fails

```c
esp_err_t esp_vfs_spiffs_unregister(const char *partition_label)
```

Unregister and unmount SPIFFS from VFS

- 参数 partition_label – Same label as passed to esp_vfs_spiffs_register.
- 返回 ESP_OK if successful

运行以下命令，将镜像烧录到 ESP32（偏移量：0x110000）：

```
python esptool.py --chip esp32 --port [port] --baud [baud] write_flash -z 0x110000_→spiffs.bin
```
Chapter 2. API

bool esp_spiffs_mounted(const char *partition_label)
Check if SPIFFS is mounted
参数 partition_label – Optional, label of the partition to check. If not specified, first partition with subtype=spiffs is used.
返回
• true if mounted
• false if not mounted

esp_err_t esp_spiffs_format (const char *partition_label)
Format the SPIFFS partition
参数 partition_label – Same label as passed to esp_vfs_spiffs_register.
返回
• ESP_OK if successful
• ESP_FAIL on error

esp_err_t esp_spiffs_info (const char *partition_label, size_t *total_bytes, size_t *used_bytes)
Get information for SPIFFS
参数
• partition_label – Same label as passed to esp_vfs_spiffs_register
• total_bytes – [out] Size of the file system
• used_bytes – [out] Current used bytes in the file system
返回
• ESP_OK if success
• ESP_ERR_INVALID_STATE if not mounted

esp_err_t esp_spiffs_check (const char *partition_label)
Check integrity of SPIFFS
参数 partition_label – Same label as passed to esp_vfs_spiffs_register
返回
• ESP_OK if successful
• ESP_ERR_INVALID_STATE if not mounted
• ESP_FAIL on error

esp_err_t esp_spiffs_gc (const char *partition_label, size_t size_to_gc)
Perform garbage collection in SPIFFS partition.
Call this function to run GC and ensure that at least the given amount of space is available in the partition. This function will fail with ESP_ERR_NOT_FINISHED if it is not possible to reclaim the requested space (that is, not enough free or deleted pages in the filesystem). This function will also fail if it fails to reclaim the requested space after CONFIG_SPIFFS_GC_MAX_RUNS number of GC iterations. On one GC iteration, SPIFFS will erase one logical block (4kB). Therefore the value of CONFIG_SPIFFS_GC_MAX_RUNS should be set at least to the maximum expected size_to_gc, divided by 4096. For example, if the application expects to make room for a 1MB file and calls esp_spiffs_gc(label, 1024 * 1024), CONFIG_SPIFFS_GC_MAX_RUNS should be set to at least 256. On the other hand, increasing CONFIG_SPIFFS_GC_MAX_RUNS value increases the maximum amount of time for which any SPIFFS GC or write operation may potentially block.
参数
• partition_label – Label of the partition to be garbage-collected. The partition must be already mounted.
• size_to_gc – The number of bytes that the GC process should attempt to make available.
返回
• ESP_OK on success
• ESP_ERR_NOT_FINISHED if GC fails to reclaim the size given by size_to_gc
• ESP_ERR_INVALID_STATE if the partition is not mounted
• ESP_FAIL on all other errors
### Structures

**struct esp_vfs_spiffs_conf_t**

Configuration structure for esp_vfs_spiffs_register.

#### Public Members

- **const char* base_path**
  
  File path prefix associated with the filesystem.

- **const char* partition_label**
  
  Optional, label of SPIFFS partition to use. If set to NULL, first partition with subtype=spiffs will be used.

- **size_t max_files**
  
  Maximum files that could be open at the same time.

- **bool format_if_mount_failed**
  
  If true, it will format the file system if it fails to mount.

### 2.9.9 虚拟文件系统组件

#### 概述

虚拟文件系统 (VFS) 组件为驱动程序提供一个统一接口，可以操作类文件对象。这类驱动程序可以是 FAT、SPIFFS 等真实文件系统，也可以是提供文件类接口的设备驱动程序。

VFS 组件支持 C 库函数（如 fopen 和 fprintf 等）与文件系统 (FS) 驱动程序协同工作。在高层级，每个 FS 驱动程序均与某些路径分属相关联。当一个 C 库函数需要打开文件时，VFS 组件将搜索与该文件所在文件路径相关联的 FS 驱动程序，并将调用传递给该驱动程序。针对该文件的读取、写入等其他操作的调用也将传递给该驱动程序。

例如，您可以使用 /fat 前缀注册 FAT 文件系统驱动，之后即可调用 fopen("/fat/file.txt", "w")。之后，VFS 将调用 FAT 驱动的 open 函数，并将参数 /file.txt 和合适的打开模式传递给 open 函数；后续对返回的 FILE* 数据流调用 C 库函数也同样会传递给 FAT 驱动。

#### 注册 FS 驱动程序

如需注册 FS 驱动程序，应用程序首先要定义一个 esp_vfs_t 结构体实例，并用指向 FS API 的函数指针填充它。

```c
esp_vfs_t myfs = {   
    .flags = ESP_VFS_FLAG_DEFAULT, 
    .write = &myfs_write, 
    .open = &myfs_open, 
    .fstat = &myfs_fstat, 
    .close = &myfs_close, 
    .read = &myfs_read, 
};
ESP_ERROR_CHECK(esp_vfs_register("/data", &myfs, NULL));
```
在上述代码中需要用到 read, write 或 read_p, write_p, 具体使用哪组函数由 FS 驱动程序 API 的声明方式决定。

示例 1: 声明 API 函数时不带额外的上下文指针参数, 即 FS 驱动程序为单例模式, 此时使用 write

```c
ssize_t myfs_write(int fd, const void *data, size_t size);
// In definition of esp_vfs_t:
    .flags = ESP_VFS_FLAG_DEFAULT,
    .write = &myfs_write,
// ... other members initialized
// When registering FS, context pointer (third argument) is NULL:
ESP_ERROR_CHECK(esp_vfs_register("/data", &myfs, NULL));
```

示例 2: 声明 API 函数时需要一个额外的上下文指针作为参数, 即可支持多个 FS 驱动程序实例, 此时使用 write_p

```c
ssize_t myfs_write(myfs_t *fs, int fd, const void *data, size_t size);
// In definition of esp_vfs_t:
    .flags = ESP_VFS_FLAG_CONTEXT_PTR,
    .write_p = &myfs_write,
// ... other members initialized
// (hypothetical myfs_mount function is used for illustrative purposes)
myfs_t* myfs_inst1 = myfs_mount(partition1->offset, partition1->size);
ESP_ERROR_CHECK(esp_vfs_register("/data1", &myfs, myfs_inst1));
// Can register another instance:
myfs_t* myfs_inst2 = myfs_mount(partition2->offset, partition2->size);
ESP_ERROR_CHECK(esp_vfs_register("/data2", &myfs, myfs_inst2));
```

同步输入/输出多路复用 VFS 组件支持通过 select() 进行同步输入/输出多路复用，其实现方式如下：
1. 调用 select(), 使用时提供的文件描述符可以属于不同的 VFS 驱动。
2. 文件描述符被分为几组，每组属于一个 VFS 驱动。
3. 非套接字 VFS 驱动的文件描述符由 start_select() 移交给指定的 VFS 驱动，后文会对此进行详述。该函数代表指定驱动 select() 的实现。这是一个非阻塞的调用，意味着在设置好检查与指定文件描述符相关事件的环境后，该函数应该立即返回。
4. 套接字 VFS 驱动的文件描述符由 socket_select() 移交给套接字 VFS 驱动，后文会对此进行详述。这是一个阻塞调用，意味着只有当有一个与套接字文件描述符相关的事件或非套接字驱动发出信号让 socket_select() 退出时，它才会返回。
5. 从各个 VFS 驱动程序收集结果，并通过对事件检查环境取消初始化来终止所有驱动程序。
6. select() 调用结束并返回适当的结果。

非套接字 VFS 驱动 如果要使用非套接字 VFS 驱动的文件描述符调用 select(), 那么需要用函数 start_select() 和 end_select() 注册该驱动，具体如下：

```c
// In definition of esp_vfs_t:
    .start_select = &uart_start_select,
    .end_select = &uart_end_select,
// ... other members initialized
```

调用 start_select() 函数可以设置环境，检测指定 VFS 驱动的文件描述符读取/写入/错误条件。
调用 end_select() 函数可以终止/取消初始化/释放由 start_select() 设置的环境。
备注：在少数情况下，在调用 end_select() 之前可能并没有调用过 start_select()。因此
end_select() 的实现必须在该情况下返回错误而不能崩溃。

如需获取更多信息，请参考 vfs/vfs_uart.c 中 UART 外设的 VFS 驱动。尤其是函数
esp_vfs_dev_uart_register()，uart_start_select() 和 uart_end_select()。

请参考以下示例，查看如何使用 VFS 文件描述符调用 select()。

- peripherals/uart/uart_select
- system/select

套接字 VFS 驱动
套接字 VFS 驱动会使用自实现的 socket_select() 函数，在读取/写入/错误条件时，非套接字 VFS 驱动会通知该函数。

可通过对定义以下函数注册套接字 VFS 驱动：

```c
// In definition of esp_vfs_t:
.socket_select = &lwip_select,
.get_socket_select_semaphore = &lwip_get_socket_select_semaphore,
.stop_socket_select = &lwip_stop_socket_select,
.stop_socket_select_isr = &lwip_stop_socket_select_isr,
// ... other members initialized
```

函数 socket_select() 是套接字驱动对 select() 的内部实现。该函数只对套接字 VFS 驱动的文件
描述符起作用。

get_socket_select_semaphore() 返回信号对象 (semaphore)，用于非套接字驱动程序中，以终止
socket_select() 的等待。

stop_socket_select() 通过传递 get_socket_select_semaphore() 函数返回的对象来终止
socket_select() 函数的等待。

stop_socket_select_isr() 与 stop_socket_select() 的作用相似，但是前者可在 ISR 中使用。

请参考 lwip/port/esp32/vfs_lwip.c 以了解使用 LWIP 的套接字驱动参考实现。

备注：如果 select() 用于套接字文件描述符，您可以禁用 CONFIG_VFS_SUPPORT_SELECT 选项来减少
代码量，提高性能。不要在 select() 调用过程中更改套接字驱动，否则会出现一些未定义行为。

路径

已注册的 FS 驱动程序均有一个路径前缀与之关联，此路径前缀即为分区的挂载点。

如果挂载点中嵌套了其他挂载点，则在打开文件时使用具有最长匹配路径前缀的挂载点。例如，假设以下文件系统已在 VFS 中注册：

- 在 /data 下注册 FS 驱动程序 1
- 在 /data/static 下注册 FS 驱动程序 2

那么：

- 打开 /data/log.txt 会调用驱动程序 FS 1；
- 打开 /data/static/index.html 需调用 FS 驱动程序 2；
- 即使 FS 驱动程序 2 中没有 /index.html，也不会在 FS 驱动程序 1 中查找 /static/index.
  html。

挂载点名称必须以路径分隔符 (/) 开头，且分隔符后至少包含一个字符。但在以下情况中，VFS 同样支
持空的挂载点名称：1. 应用程序需要提供一个”下使用的文件系统；2. 应用程序需要同时覆
盖 VFS 功能。如果没有与路径匹配的前缀，就会使用到这种文件系统。
Chapter 2. API 参考

VFS 不会对路径中的点(\.)进行特殊处理，也不会将 .. 视为对父目录的引用。在上述示例中，使用
/data/static/./log.txt 路径不会调用 FS 驱动程序 1 打开 /log.txt。特定的 FS 驱动程序 (如
FATFS) 可以以不同的方式处理文件名中的点。

执行打开文件操作时，FS 驱动程序仅得到文件的相对路径 (挂载点前缀已经被去除):

1. 以 /data 为路径前缀注册 myfs 驱动;
2. 应用程序使用 fopen(“/data/config.json”, ...)；
3. VFS 调用 myfs_open(“/config.json”, ...);
4. myfs 驱动打开 /config.json 文件。

VFS 对文件路径长度没有限制，但文件系统路径前缀受 ESP_VFS_PATH_MAX 限制，即路径前缀上限为
ESP_VFS_PATH_MAX。各个文件系统驱动则可能会对文件的文件名长度设置一些限制。

文件描述符

文件描述符是一组很小的正整数，从 0 到 FD_SETSIZE - 1，FD_SETSIZE 在 newlib sys/types.h
中定义。最大文件描述符由 CONFIG_LWIP_MAX_SOCKETS 定义。且为套接字保留。VFS 中包含一个名为
s_fd_table 的查找表，用于将全局文件描述符映射至 s_vfs 数组中注册的 VFS 驱动索引。

标准 IO 流 (stdin, stdout, stderr)

如果 menuconfig 中 UART for console output 选项没有设置为 None, 则 stdin, stdout 和 stderr
将默认从 UART 读取或写入。UART0 或 UART1 可用作标准 IO。默认情况下，UART0 使用 115200 波特率，
TX 管脚为 GPIO1, RX 管脚为 GPIO3。您可以在 menuconfig 中更改上述参数。

对 stdout 或 stderr 执行写入操作将会向 UART 发送 FIFO 发送字符，对 stdin 执行读取操作则会从
UART 接收 FIFO 中取出字符。

默认情况下，VFS 使用简单的函数对 UART 进行读写操作。在所有数据放进 UART FIFO 之前，写操作
将处于 busy-wait 状态，读操作处于非阻塞状态。只返回 FIFO 中已有数据，由于读操作为非阻塞，高层级
C 库函数调用 (如 fscanf(“%d\n", &var);) 可能获取不到所需结果。

如果应用程序使用 UART 驱动，则可以调用 esp_vfs_dev_uart_use_driver 函数来指导 VFS 使用
驱动中断、读写阻塞功能等。您也可以调用 esp_vfs_dev_uart_use_nonblocking 来恢复非阻塞
函数。

VFS 还为输入和输出提供换行符转换功能 (可选)。多数应用程序在程序内部发送或接收
以 LF (’\n’) 结尾的行，但不同的终端程序可能需要不同的换行符，比如 CR 或 CRLF，
应用程序可以通过 menuconfig 或者调用 esp_vfs_dev_uart_port_set_rx_line_endings 和
esp_vfs_dev_uart_port_set_tx_line_endings 为输入输出配置换行符。

标准流和 FreeRTOS 任务 stdin, stdout 和 stderr 的 FILE 对象在所有 FreeRTOS 任务之间共享，
指向这些对象的指针分别存储在每个任务的 struct _reent 中。

预处理器把如下代码解释为 fprintf(__getreent()->_stderr, “42\n”);

```c
fprintf(stderr, "42\n");
```

其中 __getreent () 函数将为每个任务返回一个指向 newlib libc 中 struct _reent 的指针。每个
任务的 TCB 均拥有一个 struct _reent 结构体。任务初始化后，struct _reent 结构体中的
_stdtin, _stdout 和 _stderr 将会被赋予 _GLOBAL_REENT 中 Stdin, Stdout 和 Stderr 的值，
_GLOBAL_REENT 即为 FreeRTOS 启动之后所用结构体。

这样设计带来的结果是:

- 允许设置给定任务的 stdin, stdout 和 stderr，而不影响其他任务，如通过 stdin =
  fopen(”/dev/uart/1", ”r”);
- 但使用 fclose 关闭默认 stdin, stdout 或 stderr 将同时关闭相应的 FILE 流对象，因此会影
  响其他任务;

---

Espressif Systems 1733 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
Event fds

eventfd() 是一个很强大的工具，可以循环通知基于 select() 的自定义事件。在 ESP-IDF 中，
eventfd() 的实现和 man(2) eventfd 中的描述相同，主要区别如下：

- 在调用 eventfd() 之前必须先调用 esp_vfs_eventfd_register();
- 标志中没有 EFD_CLOEXEC, EFD_NONBLOCK 和 EFD_SEMAPHORE 选项；
- EFD_SUPPORT_ISR 选项已经被添加到标志中。在中断处理程序中读取和写入 eventfd 需要这个标志。

注意，用 EFD_SUPPORT_ISR 创建 eventfd 将导致在读取、写入文件时，以及在设置这个文件的 select() 开始和结束时，暂时禁用中断。

API 参考

Header File

- components/vfs/include/esp_vfs.h

Functions

ssize_t esp_vfs_write (struct _reent *r, int fd, const void *data, size_t size)

These functions are to be used in newlib syscall table. They will be called by newlib when it needs to use any of the syscalls.

off_t esp_vfs_lseek (struct _reent *r, int fd, off_t size, int mode)

ssize_t esp_vfs_read (struct _reent *r, int fd, void *dst, size_t size)

int esp_vfs_open (struct _reent *r, const char *path, int flags, int mode)

int esp_vfs_close (struct _reent *r, int fd)

int esp_vfs_fstat (struct _reent *r, int fd, struct stat *st)

int esp_vfs_stat (struct _reent *r, const char *path, struct stat *st)

int esp_vfs_link (struct _reent *r, const char *n1, const char *n2)

int esp_vfs_unlink (struct _reent *r, const char *path)

int esp_vfs_rename (struct _reent *r, const char *src, const char *dst)

int esp_vfs_utime (const char *path, const struct utimbuf *times)

esp_err_t esp_vfs_register (const char *base_path, const esp_vfs_t *vfs, void *ctx)

Register a virtual filesystem for given path prefix.

参数

- **base_path** – file path prefix associated with the filesystem. Must be a zero-terminated C string, may be empty. If not empty, must be up to ESP_VFS_PATH_MAX characters long, and at least 2 characters long. Name must start with a “/” and must not end with “/”. For example, “/data” or “/dev/spi” are valid. These VFSes would then be called to handle file paths such as “/data/myfile.txt” or “/dev/spi/0”. In the special case of an empty base_path, a “fallback” VFS is registered. Such VFS will handle paths which are not matched by any other registered VFS.

- **vfs** – Pointer to esp_vfs_t, a structure which maps syscalls to the filesystem driver functions. VFS component doesn’t assume ownership of this pointer.
• ctx – If vfs->flags has ESP_VFS_FLAG_CONTEXT_PTR set, a pointer which should be passed to VFS functions. Otherwise, NULL.

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered.

```c
esp_err_t esp_vfs_register_fd_range(const esp_vfs_t *vfs, void *ctx, int min_fd, int max_fd)
```

Special case function for registering a VFS that uses a method other than open() to open new file descriptors from the interval <min_fd; max_fd).

This is a special-purpose function intended for registering LWIP sockets to VFS.

参数
- vfs – Pointer to esp_vfs_t. Meaning is the same as for esp_vfs_register().
- ctx – Pointer to context structure. Meaning is the same as for esp_vfs_register().
- min_fd – The smallest file descriptor this VFS will use.
- max_fd – Upper boundary for file descriptors this VFS will use (the biggest file descriptor plus one).

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered, ESP_ERR_INVALID_ARG if the file descriptor boundaries are incorrect.

```c
esp_err_t esp_vfs_register_with_id(const esp_vfs_t *vfs, void *ctx, esp_vfs_id_t *vfs_id)
```

Special case function for registering a VFS that uses a method other than open() to open new file descriptors. In comparison with esp_vfs_register_fd_range, this function doesn’t pre-registers an interval of file descriptors. File descriptors can be registered later, by using esp_vfs_register_fd.

参数
- vfs – Pointer to esp_vfs_t. Meaning is the same as for esp_vfs_register().
- ctx – Pointer to context structure. Meaning is the same as for esp_vfs_register().
- vfs_id – Here will be written the VFS ID which can be passed to esp_vfs_register_fd for registering file descriptors.

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered, ESP_ERR_INVALID_ARG if the file descriptor boundaries are incorrect.

```c
esp_err_t esp_vfs_unregister(const char* base_path)
```

Unregister a virtual filesystem for given path prefix

参数 base_path – file prefix previously used in esp_vfs_register call

返回 ESP_OK if successful, ESP_ERR_INVALID_STATE if VFS for given prefix hasn’t been registered.

```c
esp_err_t esp_vfs_unregister_with_id(esp_vfs_id_t vfs_id)
```

Unregister a virtual filesystem with the given index

参数 vfs_id – The VFS ID returned by esp_vfs_register_with_id

返回 ESP_OK if successful, ESP_ERR_INVALID_STATE if VFS for the given index hasn’t been registered.

```c
esp_err_t esp_vfs_register_fd(esp_vfs_id_t vfs_id, int*fd)
```

Special function for registering another file descriptor for a VFS registered by esp_vfs_register_with_id.

参数
- vfs_id – VFS identifier returned by esp_vfs_register_with_id.
- fd – The registered file descriptor will be written to this address.

返回 ESP_OK if the registration is successful, ESP_ERR_NO_MEM if too many file descriptors are registered, ESP_ERR_INVALID_ARG if the arguments are incorrect.

```c
esp_err_t esp_vfs_register_fd_with_local_fd(esp_vfs_id_t vfs_id, int local_fd, bool permanent, int *fd)
```

Special function for registering another file descriptor with given local_fd for a VFS registered by esp_vfs_register_with_id.

参数
- vfs_id – VFS identifier returned by esp_vfs_register_with_id.
- local_fd – The fd in the local vfs. Passing -1 will set the local fd as the (*fd) value.
• **permanent** – Whether the fd should be treated as permnet (not removed after close())
• **fd** – The registered file descriptor will be written to this address.

返回 ESP_OK if the registration is successful, ESP_ERR_NO_MEM if too many file descriptors are registered, ESP_ERR_INVALID_ARG if the arguments are incorrect.

```c
esp_err_t esp_vfs_unregister_fd (esp_vfs_id_t vfs_id, int fd)
```

Special function for unregistering a file descriptor belonging to a VFS registered by esp_vfs_register_with_id.

参数

• **vfs_id** – VFS identificator returned by esp_vfs_register_with_id.
• **fd** – File descriptor which should be unregistered.

返回 ESP_OK if the registration is successful, ESP_ERR_INVALID_ARG if the arguments are incorrect.

```c
int esp_vfs_select (int nfds, fd_set* readfds, fd_set* writefds, fd_set* errorfds, struct timeval* timeout)
```

Synchronous I/O multiplexing which implements the functionality of POSIX select() for VFS.

参数

• **nfds** – Specifies the range of descriptors which should be checked. The first nfds descriptors will be checked in each set.
• **readfds** – If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for being ready to read, and on output indicates which descriptors are ready to read.
• **writefds** – If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for being ready to write, and on output indicates which descriptors are ready to write.
• **errorfds** – If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for error conditions, and on output indicates which descriptors have error conditions.
• **timeout** – If not NULL, then points to timeval structure which specifies the time period after which the functions should time-out and return. If it is NULL, then the function will not time-out. Note that the timeout period is rounded up to the system tick and incremented by one.

返回 The number of descriptors set in the descriptor sets, or -1 when an error (specified by errno) have occurred.

```c
void esp_vfs_select_triggered (esp_vfs_select_sem_t sem)
```

Notification from a VFS driver about a read/write/error condition.

This function is called when the VFS driver detects a read/write/error condition as it was requested by the previous call to start_select.

参数 **sem** – semaphore structure which was passed to the driver by the start_select call

```c
void esp_vfs_select_triggered_isr (esp_vfs_select_sem_t sem, BaseType_t*woken)
```

Notification from a VFS driver about a read/write/error condition (ISR version)

This function is called when the VFS driver detects a read/write/error condition as it was requested by the previous call to start_select.

参数

• **sem** – semaphore structure which was passed to the driver by the start_select call
• **woken** – is set to pdTRUE if the function wakes up a task with higher priority

```c
ssize_t esp_vfs_pread (int fd, void *dst, size_t size, off_t offset)
```

Implements the VFS layer of POSIX pread()

参数

• **fd** – File descriptor used for read
• **dst** – Pointer to the buffer where the output will be written
• **size** – Number of bytes to be read
• **offset** – Starting offset of the read
A positive return value indicates the number of bytes read. -1 is return on failure and errno is set accordingly.

```c
ssize_t esp_vfs_pwrite (int fd, const void *src, size_t size, off_t offset)
```

Implements the VFS layer of POSIX pwrite()

**Parameters**
- **fd** - File descriptor used for write
- **src** - Pointer to the buffer from where the output will be read
- **size** - Number of bytes to write
- **offset** - Starting offset of the write

A positive return value indicates the number of bytes written. -1 is return on failure and errno is set accordingly.

**Structures**

```c
struct esp_vfs_select_sem_t
```

VFS semaphore type for select()

**Public Members**

```c
bool is_sem_local
```

Type of “sem” is SemaphoreHandle_t when true, defined by socket driver otherwise

```c
void *sem
```

Semaphore instance

```c
struct esp_vfs_t
```

VFS definition structure.

This structure should be filled with pointers to corresponding FS driver functions.

VFS component will translate all FDs so that the filesystem implementation sees them starting at zero. The caller sees a global FD which is prefixed with an pre-filesystem-implementation.

Some FS implementations expect some state (e.g. pointer to some structure) to be passed in as a first argument. For these implementations, populate the members of this structure which have _p suffix, set flags member to ESP_VFS_FLAG_CONTEXT_PTR and provide the context pointer to esp_vfs_register function. If the implementation doesn’t use this extra argument, populate the members without _p suffix and set flags member to ESP_VFS_FLAG_DEFAULT.

If the FS driver doesn’t provide some of the functions, set corresponding members to NULL.

**Public Members**

```c
int flags
```

ESP_VFS_FLAG_CONTEXT_PTR or ESP_VFS_FLAG_DEFAULT

```c
ssize_t (*write_p)(void *p, int fd, const void *data, size_t size)
```

Write with context pointer

```c
ssize_t (*write)(int fd, const void *data, size_t size)
```

Write without context pointer
Chapter 2. API 参考

off_t (*lseek_p)(void *p, int fd, off_t size, int mode)
   Seek with context pointer

off_t (*lseek)(int fd, off_t size, int mode)
   Seek without context pointer

ssize_t (*read_p)(void *ctx, int fd, void *dst, size_t size)
   Read with context pointer

ssize_t (*read)(int fd, void *dst, size_t size)
   Read without context pointer

ssize_t (*pread_p)(void *ctx, int fd, void *dst, size_t size, off_t offset)
   pread with context pointer

ssize_t (*pread)(int fd, void *dst, size_t size, off_t offset)
   pread without context pointer

ssize_t (*pwrite_p)(void *ctx, int fd, const void *src, size_t size, off_t offset)
   pwrite with context pointer

ssize_t (*pwrite)(int fd, const void *src, size_t size, off_t offset)
   pwrite without context pointer

int (*open_p)(void *ctx, const char *path, int flags, int mode)
   open with context pointer

int (*open)(const char *path, int flags, int mode)
   open without context pointer

int (*close_p)(void *ctx, int fd)
   close with context pointer

int (*close)(int fd)
   close without context pointer

int (*fstat_p)(void *ctx, int fd, struct stat *st)
   fstat with context pointer

int (*fstat)(int fd, struct stat *st)
   fstat without context pointer

int (*stat_p)(void *ctx, const char *path, struct stat *st)
   stat with context pointer

int (*stat)(const char *path, struct stat *st)
   stat without context pointer
int (*link_p)(void *ctx, const char *n1, const char *n2)
    link with context pointer

int (*link)(const char *n1, const char *n2)
    link without context pointer

int (*unlink_p)(void *ctx, const char *path)
    unlink with context pointer

int (*unlink)(const char *path)
    unlink without context pointer

int (*rename_p)(void *ctx, const char *src, const char *dst)
    rename with context pointer

int (*rename)(const char *src, const char *dst)
    rename without context pointer

DIR *(*opendir_p)(void *ctx, const char *name)
    opendir with context pointer

DIR *(*opendir)(const char *name)
    opendir without context pointer

struct dirent *(*readdir_p)(void *ctx, DIR *pdir)
    readdir with context pointer

struct dirent *(*readdir)(DIR *pdir)
    readdir without context pointer

int *(*readdir_r_p)(void *ctx, DIR *pdir, struct dirent *entry, struct dirent **out_dirent)
    readdir_r with context pointer

int *(*readdir_r)(DIR *pdir, struct dirent *entry, struct dirent **out_dirent)
    readdir_r without context pointer

long (*telldir_p)(void *ctx, DIR *pdir)
    telldir with context pointer

long (*telldir)(DIR *pdir)
    telldir without context pointer

void (*seekdir_p)(void *ctx, DIR *pdir, long offset)
    seekdir with context pointer

void (*seekdir)(DIR *pdir, long offset)
    seekdir without context pointer
int (*closedir_p)(void *ctx, DIR *pdir)
closedir with context pointer

int (*closedir)(DIR *pdir)
closedir without context pointer

int (*mkdir_p)(void *ctx, const char *name, mode_t mode)
mkdir with context pointer

int (*mkdir)(const char *name, mode_t mode)
mkdir without context pointer

int (*rmdir_p)(void *ctx, const char *name)
rmdir with context pointer

int (*rmdir)(const char *name)
rmdir without context pointer

int (*fcntl_p)(void *ctx, int fd, int cmd, int arg)
fcntl with context pointer

int (*fcntl)(int fd, int cmd, int arg)
fcntl without context pointer

int (*ioctl_p)(void *ctx, int fd, int cmd, va_list args)
ioctl with context pointer

int (*ioctl)(int fd, int cmd, va_list args)
ioctl without context pointer

int (*fsync_p)(void *ctx, int fd)
fsync with context pointer

int (*fsync)(int fd)
fsync without context pointer

int (*access_p)(void *ctx, const char *path, int amode)
access with context pointer

int (*access)(const char *path, int amode)
access without context pointer

int (*truncate_p)(void *ctx, const char *path, off_t length)
truncate with context pointer

int (*truncate)(const char *path, off_t length)
truncate without context pointer
int (*ftruncate_p)(void *ctx, int fd, off_t length)
  ftruncate with context pointer

int (*ftruncate)(int fd, off_t length)
  ftruncate without context pointer

int (*utime_p)(void *ctx, const char *path, const struct utimbuf *times)
  utime with context pointer

int (*utime)(const char *path, const struct utimbuf *times)
  utime without context pointer

int (*tcsetattr_p)(void *ctx, int fd, int optional_actions, const struct termios *p)
  tcsetattr with context pointer

int (*tcsetattr)(int fd, int optional_actions, const struct termios *p)
  tcsetattr without context pointer

int (*tcgetattr_p)(void *ctx, int fd, struct termios *p)
  tcgetattr with context pointer

int (*tcgetattr)(int fd, struct termios *p)
  tcgetattr without context pointer

int (*tcdrain_p)(void *ctx, int fd)
  tcdrain with context pointer

int (*tcdrain)(int fd)
  tcdrain without context pointer

int (*tcflush_p)(void *ctx, int fd, int select)
  tcflush with context pointer

int (*tcflush)(int fd, int select)
  tcflush without context pointer

int (*tcflow_p)(void *ctx, int fd, int action)
  tcflow with context pointer

int (*tcflow)(int fd, int action)
  tcflow without context pointer

pid_t (*tcgetsid_p)(void *ctx, int fd)
  tcgetsid with context pointer

pid_t (*tcgetsid)(int fd)
  tcgetsid without context pointer
int (*`tcsendbreak_p`)(void *ctx, int fd, int duration)
    tcsendbreak with context pointer

int (*`tcsendbreak`)(int fd, int duration)
    tcsendbreak without context pointer

*esp_err_t* (*`start_select`*)(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
    esp_vfs_select_sem_t sem, void **end_select_args)
    start_select is called for setting up synchronous I/O multiplexing of the desired file descriptors in the
given VFS

int (*`socket_select`)(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *
    timeout)
    socket select function for socket FDs with the functionality of POSIX select(); this should be set only for
the socket VFS

void (*`stop_socket_select`)(void *sem)
    called by VFS to interrupt the socket_select call when select is activated from a non-socket VFS driver;
set only for the socket driver

void (*`stop_socket_select_isr`)(void *sem, BaseType_t *woken)
    stop_socket_select which can be called from ISR; set only for the socket driver

void (*`get_socket_select_semaphore`)(void)
    end_select is called to stop the I/O multiplexing and deinitialize the environment created by start_select
for the given VFS

*esp_err_t* (*`end_select`)(void *end_select_args)
    get_socket_select_semaphore returns semaphore allocated in the socket driver; set only for the socket
driver

Macros

**`MAX_FDS`**
    Maximum number of (global) file descriptors.

**`ESP_VFS_PATH_MAX`**
    Maximum length of path prefix (not including zero terminator)

**`ESP_VFS_FLAG_DEFAULT`**
    Default value of flags member in esp_vfs_t structure.

**`ESP_VFS_FLAG_CONTEXT_PTR`**
    Flag which indicates that FS needs extra context pointer in syscalls.

Type Definitions

typedef int `esp_vfs_id_t`

Header File

- components/vfs/include/esp_vfs_dev.h
Functions

void esp_vfs_dev_uart_register (void)
   add /dev/uart virtual filesystem driver

   This function is called from startup code to enable serial output

void esp_vfs_dev_uart_set_rx_line_endings (esp_line_endings_t mode)
   Set the line endings expected to be received on UART.

   This specifies the conversion between line endings received on UART and newlines (’
   ’, LF) passed into stdin:

   • ESP_LINE_ENDINGS_CRLF: convert CRLF to LF
   • ESP_LINE_ENDINGS_CR: convert CR to LF
   • ESP_LINE_ENDINGS_LF: no modification

备注: this function is not threadsafe w.r.t. reading from UART

参数 mode - line endings expected on UART

void esp_vfs_dev_uart_set_tx_line_endings (esp_line_endings_t mode)
   Set the line endings to sent to UART.

   This specifies the conversion between newlines (’
   ’, LF) on stdout and line endings sent over UART:

   • ESP_LINE_ENDINGS_CRLF: convert LF to CRLF
   • ESP_LINE_ENDINGS_CR: convert LF to CR
   • ESP_LINE_ENDINGS_LF: no modification

备注: this function is not threadsafe w.r.t. writing to UART

参数 mode - line endings to send to UART

int esp_vfs_dev_uart_port_set_rx_line_endings (int uart_num, esp_line_endings_t mode)
   Set the line endings expected to be received on specified UART.

   This specifies the conversion between line endings received on UART and newlines (’
   ’, LF) passed into stdin:

   • ESP_LINE_ENDINGS_CRLF: convert CRLF to LF
   • ESP_LINE_ENDINGS_CR: convert CR to LF
   • ESP_LINE_ENDINGS_LF: no modification

备注: this function is not threadsafe w.r.t. reading from UART

参数
Chapter 2. API

```c
int esp_vfs_dev_uart_port_set_tx_line_endings(int uart_num, esp_line_endings_t mode)
```

Set the line endings to sent to specified UART.

This specifies the conversion between newlines (`\n`, LF) on stdout and line endings sent over UART:

- ESP_LINE_ENDINGS_CRLF: convert LF to CRLF
- ESP_LINE_ENDINGS_CR: convert LF to CR
- ESP_LINE_ENDINGS_LF: no modification

**Note:** this function is not thread safe w.r.t. writing to UART

```c
void esp_vfs_dev_uart_use_nonblocking(int uart_num)
```

set VFS to use simple functions for reading and writing UART. Read is non-blocking, write is busy waiting until TX FIFO has enough space. These functions are used by default.

**Parameters**
- `uart_num`: UART peripheral number

```c
void esp_vfs_dev_uart_use_driver(int uart_num)
```

set VFS to use UART driver for reading and writing

**Note:** application must configure UART driver before calling these functions. With these functions, read and write are blocking and interrupt-driven.

```c
void esp_vfs_usb_serial_jtag_use_driver(void)
```

set VFS to use USB-SERIAL-JTAG driver for reading and writing

**Note:** application must configure USB-SERIAL-JTAG driver before calling these functions. With these functions, read and write are blocking and interrupt-driven.

```c
void esp_vfs_usb_serial_jtag_use_nonblocking(void)
```

set VFS to use simple functions for reading and writing UART. Read is non-blocking, write is busy waiting until TX FIFO has enough space. These functions are used by default.

**Header File**
- `components/vfs/include/esp_vfs_eventfd.h`
Functions

```c
esp_err_t esp_vfs_eventfd_register(const esp_vfs_eventfd_config_t *config)
```
Registers the event vfs.

返回 ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered.

```c
esp_err_t esp_vfs_eventfd_unregister(void)
```
Unregisters the event vfs.

返回 ESP_OK if successful, ESP_ERR_INVALID_STATE if VFS for given prefix hasn’t been registered

```c
int eventfd(unsigned int initval, int flags)
```

Structures

```c
struct esp_vfs_eventfd_config_t
```
Eventfd vfs initialization settings.

Public Members

```c
size_t max_fds
```
The maximum number of eventfds supported

Macros

```c
EFD_SUPPORT_ISR
ESP_VFS_EVENTD_CONFIG_DEFAULT()
```

2.9.10 磨损均衡 API

概述

ESP32 所使用的 flash，特别是 SPI flash，多数具备扇区结构，且每个扇区仅允许有限次数的擦除/修改操作。为了避免过度使用某一扇区，乐鑫提供了磨损均衡组件，无需用户介入即可帮助用户均衡各个扇区之间的磨损。

磨损均衡组件包含了通过分区组件对外部 SPI flash 进行数据读取、写入、擦除与存储器映射相关的 API 函数。磨损均衡组件还具有软件上更高级别的 API 函数，与 FAT 文件系统 协同工作。

磨损均衡组件与 FAT 文件系统组件共用 FAT 文件系统的扇区。扇区大小为 4096 字节，是标准 flash 扇区的大小。在这种模式下，磨损均衡组件性能达到最佳，但需要在 RAM 中占用更多内存。

为了节省内存，磨损均衡组件还提供了另外两种模式，均使用 512 字节大小的扇区：

- **性能模式**：先将数据保存在 RAM 中，擦除扇区，然后将数据存储回 flash。如果设备在扇区擦写过程中突然断电，则整体扇区（4096 字节）数据将全部丢失。
- **安全模式**：数据先保存在 flash 中空余扇区，擦除扇区后，数据即存储回去。如果设备断电，上电后可立即恢复数据。

设备默认设置如下：

- 定义扇区大小为 512 字节
- 默认使用性能模式
您可以使用配置菜单更改设置。
磨损均衡组件不会将数据缓存在 RAM 中。写入和擦除函数直接修改 flash，函数返回后，flash 即完成修改。

磨损均衡访问 API

处理 flash 数据常用的 API 如下所示：

- `wl_mount` - 为指定分区挂载并初始化磨损均衡模块
- `wlUnmount` - 卸载分区并释放磨损均衡模块
- `wl_erase_range` - 擦除 flash 中指定的地址范围
- `wl_write` - 将数据写入分区
- `wl_read` - 从分区读取数据
- `wl_size` - 返回可用内存的大小（以字节为单位）
- `wl_sector_size` - 返回一个扇区的大小

请尽量避免直接使用原始磨损均衡函数，建议您使用文件系统特定的函数。

内存大小

内存大小是根据分区参数在磨损均衡模块中计算所得，由于模块使用 flash 分区扇区存储内部数据，因此计算所得内存大小有少许偏差。

另请参阅

- `FAT 文件系统`
- `分区表`

应用示例

`storage/wear_levelling` 中提供了一款磨损均衡驱动与 FatFs 库结合使用的示例。该示例初始化磨损均衡驱动，挂载 FAT 文件系统分区，并使用 POSIX（可移植操作系统接口）和 C 库 API 中写入和读取数据。如需了解更多信息，请参考 `storage/wear_levelling/README.md`。

高级 API 参考

头文件

- `fatfs/vfs/esp_vfs_fat.h`

函数

```c
esp_err_t esp_vfs_fat_spiflash_mount_rw_wl(const char* base_path, const char* partition_label,
const esp_vfs_fat_mount_config_t *mount_config,
wl_handle_t *wl_handle)
```

Convenience function to initialize FAT filesystem in SPI flash and register it in VFS.

This is an all-in-one function which does the following:

- finds the partition with defined partition_label. Partition label should be configured in the partition table.
- initializes flash wear levelling library on top of the given partition
- mounts FAT partition using FATFS library on top of flash wear levelling library
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact.
Chapter 2. API

参数
- **base_path** - path where FATFS partition should be mounted (e.g. “/spiflash”)
- **partition_label** - label of the partition which should be used
- **mount_config** - pointer to structure with extra parameters for mounting FATFS
- **wl_handle** - [out] wear levelling driver handle

返回
- ESP_OK on success
- ESP_ERR_NOT_FOUND if the partition table does not contain FATFS partition with given label
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_rw_wl was already called
- ESP_ERR_NO_MEM if memory cannot be allocated
- ESP_FAIL if partition cannot be mounted
- other error codes from wear levelling library, SPI flash driver, or FATFS drivers

结构 **esp_vfs_fat_mount_config_t**

Configuration arguments for esp_vfs_fat_sdmmc_mount and esp_vfs_fat_spiflash_mount_rw_wl functions.

**Public Members**

布尔变量 **format_if_mount_failed**

If FAT partition can not be mounted, and this parameter is true, create partition table and format the filesystem.

整型变量 **max_files**

Max number of open files.

大小数型变量 **allocation_unit_size**

If format_if_mount_failed is set, and mount fails, format the card with given allocation unit size. Must be a power of 2, between sector size and 128 * sector size. For SD cards, sector size is always 512 bytes. For wear levelling, sector size is determined by CONFIG_WL_SECTOR_SIZE option.

Using larger allocation unit size will result in higher read/write performance and higher overhead when storing small files.

Setting this field to 0 will result in allocation unit set to the sector size.

布尔变量 **disk_status_check_enable**

Enables real if_disk_status function implementation for SD cards (ff_sdmmc_status). Possibly slows down IO performance.

Try to enable if you need to handle situations when SD cards are not unmouted properly before physical removal or you are experiencing issues with SD cards.

Doesn’t do anything for other memory storage media.

错误码 **esp_err_t esp_vfs_fat_spiflash_unmount_rw_wl**(const char*base_path, wl_handle_t wl_handle)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_spiflash_mount_rw_wl.

参数
- **base_path** – path where partition should be registered (e.g. “/spiflash”)
- **wl_handle** – wear levelling driver handle returned by esp_vfs_fat_spiflash_mount_rw_wl

返回
- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount_rw_wl hasn’t been called
Chapter 2. API 参考

中层 API 参考

Header File

- components/wear_levelling/include/wear_levelling.h

Functions

```c
esp_err_t wl_mount(const esp_partition_t *partition, wl_handle_t *out_handle)
```

Mount WL for defined partition.

参数

- `partition` – that will be used for access
- `out_handle` – handle of the WL instance

返回

- ESP_OK, if the allocation was successfully;
- ESP_ERR_INVALID_ARG, if WL allocation was unsuccessful;
- ESP_ERR_NO_MEM, if there was no memory to allocate WL components;

```c
esp_err_t wlUnmount(wl_handle_t handle)
```

Unmount WL for defined partition.

参数 `handle` – WL partition handle

返回

- ESP_OK, if the operation completed successfully;
- or one of error codes from lower-level flash driver.

```c
esp_err_t wl_erase_range(wl_handle_t handle, size_t start_addr, size_t size)
```

Erase part of the WL storage.

参数

- `handle` – WL handle that are related to the partition
- `start_addr` – Address where erase operation should start. Must be aligned to the result of function `wl_sector_size(⋯)`.
- `size` – Size of the range which should be erased, in bytes. Must be divisible by result of function `wl_sector_size(⋯)`.

返回

- ESP_OK, if the range was erased successfully;
- ESP_ERR_INVALID_ARG, if iterator or dst are NULL;
- ESP_ERR_INVALID_SIZE, if erase would go out of bounds of the partition;
- or one of error codes from lower-level flash driver.

```c
esp_err_t wl_write(wl_handle_t handle, size_t dest_addr, const void *src, size_t size)
```

Write data to the WL storage.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using `wl_erase_range` function.

备注: Prior to writing to WL storage, make sure it has been erased with `wl_erase_range` call.
• ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition;
• or one of error codes from lower-level flash driver.

\textit{esp_err_t} \texttt{wl_read} (\texttt{wl_handle_t} handle, size_t src_addr, void* dest, size_t size)

Read data from the WL storage.

\begin{itemize}
  \item \texttt{handle} - WL module instance that was initialized before
  \item \texttt{dest} - Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least `size` bytes long.
  \item \texttt{src_addr} - Address of the data to be read, relative to the beginning of the partition.
  \item \texttt{size} - Size of data to be read, in bytes.
\end{itemize}

\begin{itemize}
  \item ESP_OK, if data was read successfully;
  \item ESP_ERR_INVALID_ARG, if src_offset exceeds partition size;
  \item ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition;
  \item or one of error codes from lower-level flash driver.
\end{itemize}

\texttt{size_t} \texttt{wl_size} (\texttt{wl_handle_t} handle)

Get size of the WL storage.

\begin{itemize}
  \item \texttt{handle} - WL module handle that was initialized before
  \item usable size, in bytes
\end{itemize}

\texttt{size_t} \texttt{wl_sector_size} (\texttt{wl_handle_t} handle)

Get sector size of the WL instance.

\begin{itemize}
  \item \texttt{handle} - WL module handle that was initialized before
  \item sector size, in bytes
\end{itemize}

\textbf{Macros}

\texttt{WL_INVALID_HANDLE}

\textbf{Type Definitions}

\texttt{typedef int32_t \textit{wl_handle_t}}

\texttt{wear levelling handle}

此部分 API 代码示例存放 ESP-IDF 示例项目的 storage 目录下。

\section*{2.10 System API}

\subsection*{2.10.1 App Image Format}

An application image consists of the following structures:

1. The \texttt{esp_image_header_t} structure describes the mode of SPI flash and the count of memory segments.
2. The \texttt{esp_image_segment_header_t} structure describes each segment, its length, and its location in ESP32's memory, followed by the data with a length of \texttt{data_len}. The data offset for each segment in the image is calculated in the following way:

\begin{itemize}
  \item offset for 0 Segment = sizeof(\texttt{esp_image_header_t}) + sizeof(\texttt{esp_image_segment_header_t}).
  \item offset for 1 Segment = offset for 0 Segment + length of 0 Segment + sizeof(\texttt{esp_image_segment_header_t}).
\end{itemize}
You can also see the information on segments in the ESP-IDF logs while your application is booting:

```
>> esp tool.py --chip esp32 image_info build/app.bin

esp tool.py v2.3.1
Image version: 1
Entry point: 40080ea4
13 segments
Segment 1: len 0x13ce0 load 0x3f400020 file_offs 0x00000018 SOC_DROM
Segment 2: len 0x00000 load 0x3ff80000 file_offs 0x00013d00 SOC_RTC_DRAM
Segment 3: len 0x00000 load 0x3ff80000 file_offs 0x00013d08 SOC_RTC_DRAM
Segment 4: len 0x028e0 load 0x3ff80000 file_offs 0x00013d10 DRAM
Segment 5: len 0x000000 load 0x3ff828e0 file_offs 0x000165f6 DRAM
Segment 6: len 0x00040 load 0x40080000 file_offs 0x00016600 SOC_IRAM
Segment 7: len 0x09600 load 0x40080400 file_offs 0x00016a08 SOC_IRAM
Segment 8: len 0x62e4c load 0x400d0018 file_offs 0x00020010 SOC_IRAM
Segment 9: len 0x06ecc load 0x40088a00 file_offs 0x00082e64 SOC_IROM
Segment 10: len 0x00000 load 0x400c0000 file_offs 0x000895b8 SOC_RTC_IRAM
Segment 11: len 0x000004 load 0x50000000 file_offs 0x00089560 SOC_RTC_DATA
Segment 12: len 0x00000 load 0x50000004 file_offs 0x0008956c SOC_RTC_DATA
Segment 13: len 0x00000 load 0x50000004 file_offs 0x00089b74 SOC_RTC_DATA
Checksum: e8 (valid) Validation Hash: ...
```

```
--407089ca0eae2b8b83b4120979d3354b1c938a49cb7a0c977f240474ef2ec76b (valid)
```

You can also see the information on segments in the ESP-IDF logs while your application is booting:

```
I (443) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x13ce0 (...) 81120) map
I (489) esp_image: segment 1: paddr=0x00033d08 vaddr=0x3ff80000 size=0x00000 ( 0) load
I (530) esp_image: segment 2: paddr=0x00033d10 vaddr=0x3ff80000 size=0x00000 ( 0) load
I (571) esp_image: segment 3: paddr=0x00033d18 vaddr=0x3ff80000 size=0x028e0 (...) 10464) load
I (612) esp_image: segment 4: paddr=0x00036600 vaddr=0x3ff828e0 size=0x00000 ( 0) load
I (654) esp_image: segment 5: paddr=0x00036608 vaddr=0x40080000 size=0x00040 (...) 1024) load
I (695) esp_image: segment 6: paddr=0x00036a10 vaddr=0x40080400 size=0x09600 (...) 38400) load
I (737) esp_image: segment 7: paddr=0x00040018 vaddr=0x400d0018 size=0x62e4c ...) 405068) map
I (847) esp_image: segment 8: paddr=0x000a2e6c vaddr=0x40089a00 size=0x06cec (...) 27884) load
I (888) esp_image: segment 9: paddr=0x000a9b60 vaddr=0x400c0000 size=0x00000 ( 0) load
I (929) esp_image: segment 10: paddr=0x000a9b58 vaddr=0x50000000 size=0x00004 ( 4) load
I (971) esp_image: segment 11: paddr=0x000a9b74 vaddr=0x50000004 size=0x00000 ( 0) load
I (1012) esp_image: segment 12: paddr=0x000a9b7c vaddr=0x50000004 size=0x00000 ( 0) load
```

For more details on the type of memory segments and their address ranges, see ESP32 Technical Reference Manual > System and Memory > Embedded Memory [PDF].

3. The image has a single checksum byte after the last segment. This byte is written on a sixteen byte padded
boundary, so the application image might need padding.

4. If the hash_appended field from `esp_image_header_t` is set then a SHA256 checksum will be appended. The value of SHA256 is calculated on the range from the first byte and up to this field. The length of this field is 32 bytes.

5. If the option `CONFIG_SECURE_SIGNED_APPS_SCHEME` is set to ECDSA then the application image will have an additional 68 bytes for an ECDSA signature, which includes:
   - version word (4 bytes),
   - signature data (64 bytes).

6. If the option `CONFIG_SECURE_SIGNED_APPS_SCHEME` is set to RSA or ECDSA (V2) then the application image will have an additional signature sector of 4K size. For more details on the format of this signature sector, please refer to `Signature Block Format`.

**Application Description**

The DROM segment of the application binary starts with the `esp_app_desc_t` structure which carries specific fields describing the application:

- magic_word - the magic word for the `esp_app_desc` structure.
- secure_version - see `Anti-rollback`.
- version - see `App version`.
- project_name is filled from PROJECT_NAME.*
- time and date - compile time and date.
- idf_ver - version of ESP-IDF.
- app_elf_sha256 - contains sha256 for the application ELF file.

* - The maximum length is 32 characters, including null-termination character. For example, if the length of PROJECT_NAME exceeds 31 characters, the excess characters will be disregarded.

This structure is useful for identification of images uploaded OTA because it has a fixed offset = sizeof(`esp_image_header_t`) + sizeof(`esp_image_segment_header_t`). As soon as a device receives the first fragment containing this structure, it has all the information to determine whether the update should be continued or not.

To obtain the `esp_app_desc_t` structure for the currently running application, use `esp_app_get_description()`.

To obtain the `esp_app_desc_t` structure for another OTA partition, use `esp_ota_get_partition_description()`.

**Adding a Custom Structure to an Application**

Users also have the opportunity to have similar structure with a fixed offset relative to the beginning of the image. The following pattern can be used to add a custom structure to your image:

```c
const __attribute__((section(".rodata_custom_desc"))) esp_custom_app_desc_t custom_app_desc = { ... }
```

Offset for custom structure is sizeof(`esp_image_header_t`) + sizeof(`esp_image_segment_header_t`) + sizeof(`esp_app_desc_t`).

To guarantee that the custom structure is located in the image even if it is not used, you need to add `target_link_libraries(${COMPONENT_TARGET} "-u custom_app_desc")` into CMakeLists.txt.

**API Reference**

**Header File**

- components/bootloader_support/include/esp_app_format.h
Chapter 2. API

**Structures**

```
struct esp_image_header_t
    Main header of binary image.
```

**Public Members**

uint8_t `magic`
    Magic word ESP_IMAGE_HEADER_MAGIC

uint8_t `segment_count`
    Count of memory segments

uint8_t `spi_mode`
    Flash read mode (esp_image_spi_mode_t as uint8_t)

uint8_t `spi_speed`
    Flash frequency (esp_image_spi_freq_t as uint8_t)

uint8_t `spi_size`
    Flash chip size (esp_image_flash_size_t as uint8_t)

uint32_t `entry_addr`
    Entry address

uint8_t `wp_pin`
    WP pin when SPI pins set via efuse (read by ROM bootloader, the IDF bootloader uses software to configure the WP pin and sets this field to 0xEE=disabled)

uint8_t `spi_pin_drv[3]`
    Drive settings for the SPI flash pins (read by ROM bootloader)

`esp_chip_id_t chip_id`
    Chip identification number

uint8_t `min_chip_rev`
    Minimal chip revision supported by image After the Major and Minor revision eFuses were introduced into the chips, this field is no longer used. But for compatibility reasons, we keep this field and the data in it. Use min_chip_rev_full instead. The software interprets this as a Major version for most of the chips and as a Minor version for the ESP32-C3.

uint16_t `min_chip_rev_full`
    Minimal chip revision supported by image, in format: major * 100 + minor

uint16_t `max_chip_rev_full`
    Maximal chip revision supported by image, in format: major * 100 + minor

uint8_t `reserved[4]`
    Reserved bytes in additional header space, currently unused
uint8_t hash_append

If 1, a SHA256 digest “simple hash” (of the entire image) is appended after the checksum. Included in image length. This digest is separate to secure boot and only used for detecting corruption. For secure boot signed images, the signature is appended after this (and the simple hash is included in the signed data).

struct esp_image_segment_header_t

Header of binary image segment.

Public Members

uint32_t load_addr

Address of segment

uint32_t data_len

Length of data

Macros

ESP_IMAGE_HEADER_MAGIC

The magic word for the esp_image_header_t structure.

ESP_IMAGE_MAX_SEGMENTS

Max count of segments in the image.

Enumerations

denum esp_chip_id_t

ESP chip ID.

Values:

enumerator ESP_CHIP_ID_ESP32

chip ID: ESP32

enumerator ESP_CHIP_ID_ESP32S2

chip ID: ESP32-S2

enumerator ESP_CHIP_ID_ESP32C3

chip ID: ESP32-C3

enumerator ESP_CHIP_ID_ESP32S3

chip ID: ESP32-S3

enumerator ESP_CHIP_ID_ESP32C2

chip ID: ESP32-C2

enumerator ESP_CHIP_ID_ESP32C6

chip ID: ESP32-C6
enumerator ESP_CHIP_ID_INVALID
                   Invalid chip ID (we defined it to make sure the esp_chip_id_t is 2 bytes size)

enum esp_image_spi_mode_t
    SPI flash mode, used in esp_image_header_t.
    Values:

enumerator ESP_IMAGE_SPI_MODE_QIO
    SPI mode QIO

enumerator ESP_IMAGE_SPI_MODE_QOUT
    SPI mode QOUT

enumerator ESP_IMAGE_SPI_MODE_DIO
    SPI mode DIO

enumerator ESP_IMAGE_SPI_MODE_DOUT
    SPI mode DOUT

enumerator ESP_IMAGE_SPI_MODE_FAST_READ
    SPI mode FAST_READ

enumerator ESP_IMAGE_SPI_MODE_SLOW_READ
    SPI mode SLOW_READ

enum esp_image_spi_freq_t
    SPI flash clock division factor.
    Values:

enumerator ESP_IMAGE_SPI_SPEED_DIV_2
    The SPI flash clock frequency is divided by 2 of the clock source

enumerator ESP_IMAGE_SPI_SPEED_DIV_3
    The SPI flash clock frequency is divided by 3 of the clock source

enumerator ESP_IMAGE_SPI_SPEED_DIV_4
    The SPI flash clock frequency is divided by 4 of the clock source

enumerator ESP_IMAGE_SPI_SPEED_DIV_1
    The SPI flash clock frequency equals to the clock source

enum esp_image_flash_size_t
    Supported SPI flash sizes.
    Values:

enumerator ESP_IMAGE_FLASH_SIZE_1MB
    SPI flash size 1 MB
2.10.2 Application Level Tracing

Overview

IDF provides a useful feature for program behavior analysis called Application Level Tracing. The feature can be enabled in menuconfig and allows transfer of arbitrary data between the host and ESP32 via JTAG interface with minimal overhead on program execution. Developers can use this library to send application specific state of execution to the host and receive commands or other type of info in the opposite direction at runtime. The main use cases of this library are:

1. Collecting application specific data, see 特定应用程序的跟踪
2. Lightweight logging to the host, see 记录日志到主机
3. System behaviour analysis, see 基于 SEGGER SystemView 的系统行为分析

API Reference

Header File

* components/app_trace/include/esp_app_trace.h

Functions

```c
esp_err_t esp_apptrace_init (void)
```

Initializes application tracing module.

备注：Should be called before any esp_apptrace_xxx call.
Chapter 2. API 参考

返回 ESP_OK on success, otherwise see esp_err_t

does not protect internal data by lock.

参数
- `buf` - Address of buffer to use for down channel (host to target) data.
- `size` - Size of the buffer.

uint8_t * esp_apptrace_buffer_get (esp_apptrace_dest_t dest, uint32_t size, uint32_t tmo)

Allocates buffer for trace data. Once the data in the buffer is ready to be sent, esp_apptrace_buffer_put must be called to indicate it.

参数
- `dest` - Indicates HW interface to send data.
- `size` - Size of data to write to trace buffer.
- `tmo` - Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 non-NULL on success, otherwise NULL.

esp_err_t esp_apptrace_buffer_put (esp_apptrace_dest_t dest, uint8_t *ptr, uint32_t tmo)

Indicates that the data in the buffer is ready to be sent. This function is a counterpart of and must be preceded by esp_apptrace_buffer_get.

参数
- `dest` - Indicates HW interface to send data. Should be identical to the same parameter in call to esp_apptrace_buffer_get.
- `ptr` - Address of trace buffer to release. Should be the value returned by call to esp_apptrace_buffer_get.
- `tmo` - Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

esp_err_t esp_apptrace_write (esp_apptrace_dest_t dest, const void *data, uint32_t size, uint32_t tmo)

Writes data to trace buffer.

参数
- `dest` - Indicates HW interface to send data.
- `data` - Address of data to write to trace buffer.
- `size` - Size of data to write to trace buffer.
- `tmo` - Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

int esp_apptrace_vprintf_to (esp_apptrace_dest_t dest, uint32_t tmo, const char *fmt, va_list ap)

vprintf-like function to send log messages to host via specified HW interface.

参数
- `dest` - Indicates HW interface to send data.
- `tmo` - Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.
- `fmt` - Address of format string.
- `ap` - List of arguments.

返回 Number of bytes written.
int esp_apptrace_vprintf (const char *fmt, va_list ap)

vprintf-like function to send log messages to host.

参数
  • fmt – Address of format string.
  • ap – List of arguments.
  返回 Number of bytes written.

esp_err_t esp_apptrace_flush (esp_apptrace_dest_t dest, uint32_t tmo)
Flushes remaining data in trace buffer to host.

参数
  • dest – Indicates HW interface to flush data on.
  • tmo – Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.
  返回 ESP_OK on success, otherwise see esp_err_t

esp_err_t esp_apptrace_flush_nolock (esp_apptrace_dest_t dest, uint32_t min_sz, uint32_t tmo)
Flushes remaining data in trace buffer to host without locking internal data. This is a special version of esp_apptrace_flush which should be called from panic handler.

参数
  • dest – Indicates HW interface to flush data on.
  • min_sz – Threshold for flushing data. If current filling level is above this value, data will be flushed. TRAX destinations only.
  • tmo – Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.
  返回 ESP_OK on success, otherwise see esp_err_t

esp_err_t esp_apptrace_read (esp_apptrace_dest_t dest, void *data, uint32_t size, uint32_t tmo)
Reads host data from trace buffer.

参数
  • dest – Indicates HW interface to read the data on.
  • data – Address of buffer to put data from trace buffer.
  • size – Pointer to store size of read data. Before call to this function pointed memory must hold requested size of data
  • tmo – Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.
  返回 ESP_OK on success, otherwise see esp_err_t

uint8_t *esp_apptrace_down_buffer_get (esp_apptrace_dest_t dest, uint32_t size, uint32_t tmo)
Retrieves incoming data buffer if any. Once data in the buffer is processed, esp_apptrace_down_buffer_put must be called to indicate it.

参数
  • dest – Indicates HW interface to receive data.
  • size – Address to store size of available data in down buffer. Must be initialized with requested value.
  • tmo – Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.
  返回 non-NULL on success, otherwise NULL.

esp_err_t esp_apptrace_down_buffer_put (esp_apptrace_dest_t dest, uint8_t *ptr, uint32_t tmo)
Indicates that the data in the down buffer is processed. This function is a counterpart of and must be preceded by esp_apptrace_down_buffer_get.

参数
  • dest – Indicates HW interface to receive data. Should be identical to the same parameter in call to esp_apptrace_down_buffer_get.
  • ptr – Address of trace buffer to release. Should be the value returned by call to esp_apptrace_down_buffer_get.
• tmo – Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK on success, otherwise see esp_err_t

bool esp_apprtrace_host_is_connected(esp_apprtrace_dest_t dest)
Checks whether host is connected.
参数 dest – Indicates HW interface to use.
返回 true if host is connected, otherwise false

void *esp_apprtrace_fopen(esp_apprtrace_dest_t dest, const char *path, const char *mode)
Opens file on host. This function has the same semantic as ‘fopen’ except for the first argument.
参数
• dest – Indicates HW interface to use.
• path – Path to file.
• mode – Mode string. See fopen for details.
返回 non zero file handle on success, otherwise 0

int esp_apprtrace_fclosel(esp_apprtrace_dest_t dest, void *stream)
Closes file on host. This function has the same semantic as ‘fclose’ except for the first argument.
参数
• dest – Indicates HW interface to use.
• stream – File handle returned by esp_apprtrace_fopen.
返回 Zero on success, otherwise non-zero. See fclose for details.

size_t esp_apprtrace_fwritel(esp_apprtrace_dest_t dest, const void *ptr, size_t size, size_t nmemb, void *stream)
Writes to file on host. This function has the same semantic as ‘fwrite’ except for the first argument.
参数
• dest – Indicates HW interface to use.
• ptr – Address of data to write.
• size – Size of an item.
• nmemb – Number of items to write.
• stream – File handle returned by esp_apprtrace_fopen.
返回 Number of written items. See fwrite for details.

size_t esp_apprtrace_freadl(esp_apprtrace_dest_t dest, void *ptr, size_t size, size_t nmemb, void *stream)
Read file on host. This function has the same semantic as ‘fread’ except for the first argument.
参数
• dest – Indicates HW interface to use.
• ptr – Address to store read data.
• size – Size of an item.
• nmemb – Number of items to read.
• stream – File handle returned by esp_apprtrace_fopen.
返回 Number of read items. See fread for details.

int esp_apprtrace_fseekl (esp_apprtrace_dest_t dest, void *stream, long offset, int whence)
Set position indicator in file on host. This function has the same semantic as ‘fseek’ except for the first argument.
参数
• dest – Indicates HW interface to use.
• stream – File handle returned by esp_apprtrace_fopen.
• offset – Offset. See fseek for details.
• whence – Position in file. See fseek for details.
返回 Zero on success, otherwise non-zero. See fseek for details.
int esp_apptrace_ftell (esp_apptrace_dest_t dest, void *stream)

Get current position indicator for file on host. This function has the same semantic as `ftell` except for the first argument.

参数
- dest - Indicates HW interface to use.
- stream - File handle returned by esp_apptrace_fopen.

返回 Current position in file. See ftell for details.

int esp_apptrace_fstop (esp_apptrace_dest_t dest)

Indicates to the host that all file operations are complete. This function should be called after all file operations are finished and indicate to the host that it can perform cleanup operations (close open files etc.).

参数 dest - Indicates HW interface to use.

返回 ESP_OK on success, otherwise see esp_err_t

void esp_gcov_dump (void)

Triggers gcov info dump. This function waits for the host to connect to target before dumping data.

Enumerations

enum esp_apptrace_dest_t

Application trace data destinations bits.

Values:

enumerator ESP_APPTRACE_DEST_JTAG

JTAG destination.

enumerator ESP_APPTRACE_DEST_TRAX

xxx_TRAX name is obsolete, use more common xxx_JTAG

enumerator ESP_APPTRACE_DEST_UART

UART destination.

enumerator ESP_APPTRACE_DEST_MAX

enumerator ESP_APPTRACE_DEST_NUM

Header File

- components/app_trace/include/esp_sysview_trace.h

Functions

static inline esp_err_t esp_sysview_flush (uint32_t tmo)

Flushes remaining data in SystemView trace buffer to host.

参数 tmo - Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

返回 ESP_OK.

int esp_sysview_vprintf (const char *format, va_list args)

vprintf-like function to sent log messages to the host.

参数
- format - Address of format string.
- args - List of arguments.
**Chapter 2. API 参考**

**2.10.3 Call function with external stack**

**Overview**

A given function can be executed with a user allocated stack space which is independent of current task stack, this mechanism can be used to save stack space wasted by tasks which call a common function with intensive stack usage such as printf. The given function can be called inside the shared stack space which is a callback function deferred by calling `esp_execute_shared_stack_function()`, passing that function as parameter.

**Usage**

`esp_execute_shared_stack_function()` takes four arguments:

- a mutex object allocated by the caller, which is used to protect if the same function shares its allocated stack
- a pointer to the top of stack used for that function
- the size of stack in bytes
- a pointer to the shared stack function

The user defined function will be deferred as a callback and can be called using the user allocated space without taking space from current task stack.

The usage may look like the code below:

```c
void external_stack_function(void)
{
    printf("Executing this printf from external stack! \n");
}

//Let's suppose we want to call printf using a separated stack space
//allowing the app to reduce its stack size.
void app_main()
```


```c
//Allocate a stack buffer, from heap or as a static form:
portSTACK_TYPE *shared_stack = malloc(8192 * sizeof(portSTACK_TYPE));
assert(shared_stack != NULL);

//Allocate a mutex to protect its usage:
SemaphoreHandle_t printf_lock = xSemaphoreCreateMutex();
assert(printf_lock != NULL);

//Call the desired function using the macro helper:
esp_execute_shared_stack_function(printf_lock,
         shared_stack, 8192,
         external_stack_function);

vSemaphoreDelete(printf_lock);
free(shared_stack);
```

### API Reference

**Header File**

- components/esp_system/include/esp_expression_with_stack.h

**Functions**

`void esp_execute_shared_stack_function(SemaphoreHandle_t lock, void *stack, size_t stack_size, shared_stack_function function)`

Calls user defined shared stack space function.

备注：if either lock, stack or stack size is invalid, the expression will be called using the current stack.

### 参数

- `lock` - Mutex object to protect in case of shared stack
- `stack` - Pointer to user allocated stack
- `stack_size` - Size of current stack in bytes
- `function` - Pointer to the shared stack function to be executed

### Macros

`ESP_EXECUTE_EXPRESSION_WITH_STACK(lock, stack, stack_size, expression)`

### Type Definitions

typedef void (*shared_stack_function)(void)

## 2.10.4 Chip Revision

### Overview

A new chip versioning logic was introduced in new chips. Chips have several eFuse version fields:

- Major wafer version (WAFER_VERSION_MAJOR eFuse)
- Minor wafer version (WAFER_VERSION_MINOR eFuse)
• Ignore maximal revision (DISABLE_WAFTER_VERSION_MAJOR eFusse)

The new versioning logic is being introduced to distinguish changes in chips as breaking changes and non-breaking changes. Chips with non-breaking changes can run the same software as the previous chip. The previous chip means that the major version is the same.

If the newly released chip does not have breaking changes, that means it can run the same software as the previous chip, then in that chip we keep the same major version and increment the minor version by 1. Otherwise, if there is a breaking change in the newly released chip, meaning it can not run the same software as the previous chip, then in that chip we increase the major version and set the minor version to 0.

The software supports a number of revisions, from the minimum to the maximum (the min/max configs are defined in Kconfig). If the software is unaware of a new chip (when the chip version is out of range), it will refuse to run on it unless the Ignore maximum revision restrictions bit is set. This bit removes the upper revision limit.

Minimum versions limits the software to only run on a chip revision that is high enough to support some features. Maximum version is the maximum version that is well-supported by current software. When chip version is above the maximum version, software will reject to boot, because it may not work on, or work with risk on the chip.

Adding the major and minor wafer revision make the versioning logic is branchable.

<table>
<thead>
<tr>
<th>ECO</th>
<th>Revision (Major.Minor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECO0</td>
<td>v0.0</td>
</tr>
<tr>
<td>ECO1</td>
<td>v1.0</td>
</tr>
<tr>
<td>ECO1.1</td>
<td>v1.1</td>
</tr>
<tr>
<td>ECO2</td>
<td>v2.0</td>
</tr>
<tr>
<td>ECO3</td>
<td>v3.0</td>
</tr>
<tr>
<td>ECO4</td>
<td>v3.1</td>
</tr>
</tbody>
</table>

Chip Revision vX.Y, where:
- **X** means Major wafer version. If it is changed, it means that the current software version is not compatible with this released chip and the software must be updated to use this chip.
- **Y** means Minor wafer version. If it is changed that means the current software version is compatible with the released chip, and there is no need to update the software.

The vX.Y chip version format will be used further instead of the ECO number.

### Representing Revision Requirement Of A Binary Image

The 2nd stage bootloader and the application binary images have the `esp_image_header_t` header, which stores the revision numbers of the chip on which the software can be run. This header has 3 fields related to revisions:

- **min_chip_rev** - Minimal chip MAJOR revision required by image (but for ESP32-C3 it is MINOR revision). Its value is determined by `CONFIG_ESP32_REV_MIN`.
- **min_chip_rev_full** - Minimal chip MINOR revision required by image in format: major * 100 + minor. Its value is determined by `CONFIG_ESP32_REV_MIN`.
max_chip_rev_full - Maximal chip revision required by image in format: major * 100 + minor. Its value is determined by CONFIG_ESP32_REV_MAX_FULL. It cannot be changed by user. Only Espressif can change it when a new version will be supported in IDF.

Chip Revision APIs

These APIs help to get chip revision from eFuses:

• efuse_hal_chip_revision(). It returns revision in the major * 100 + minor format.
• efuse_hal_get_major_chip_version(). It returns Major revision.
• efuse_hal_get_minor_chip_version(). It returns Minor revision.

The following Kconfig definitions (in major * 100 + minor format) that can help add the chip revision dependency to the code:

• CONFIG_ESP32_REV_MIN_FULL
• CONFIG_ESP_REV_MIN_FULL
• CONFIG_ESP32_REV_MAX_FULL
• CONFIG_ESP_REV_MAX_FULL

Maximal And Minimal Revision Restrictions

The order for checking the minimum and maximum revisions:

1. The 1st stage bootloader (ROM bootloader) does not check minimal and maximal revision fields from esp_image_header_t before running the 2nd stage bootloader.
2. The 2nd stage bootloader checks at the initialization phase that bootloader itself can be launched on the chip of this revision. It extracts the minimum revision from the header of the bootloader image and checks against the chip revision from eFuses. If the chip revision is less than the minimum revision, the bootloader refuses to boot up and aborts. The maximum revision is not checked at this phase.
3. Then the 2nd stage bootloader checks the revision requirements of the application. It extracts the minimum and maximum revisions from the header of the application image and checks against the chip revision from eFuses. If the chip revision is less than the minimum revision or higher than the maximum revision, the bootloader refuses to boot up and aborts. However, if the Ignore maximal revision bit is set, the maximum revision constraint can be ignored. The ignore bit is set by the customer themselves when there is confirmation that the software is able to work with this chip revision.
4. Further, at the OTA update stage, the running application checks if the new software matches the chip revision. It extracts the minimum and maximum revisions from the header of the new application image and checks against the chip revision from eFuses. It checks for revision matching in the same way that the bootloader does, so that the chip revision is between the min and max revisions (logic of ignoring max revision also applies).

Issues

1. If the 2nd stage bootloader is run on the chip revision < minimum revision shown in the image, a reboot occurs. The following message will be printed:

   Image requires chip rev >= v3.0, but chip is v1.0

To resolve this issue:

• make sure the chip you are using is suitable for the software, or use a chip with the required minimum revision or higher.
• update the software with CONFIG_ESP32_REV_MIN to get it <= the revision of chip being used

2. If application does not match minimal and maximal chip revisions, a reboot occurs. The following message will be printed:
To resolve this issue, update the IDF to a newer version that supports the used chip (CONFIG_ESP32_REV_MAX_FULL). Another way to fix this is to set the Ignore maximal revision bit in eFuse or use a chip that is suitable for the software.

**Backward Compatible With Bootloaders Built By Older ESP-IDF Versions**

The old bootloaders (IDF < 5.0) do not know about Major and Minor wafer version eFuses. They use one single eFuse for this - wafer version.

The old bootloaders did not read the minor wafer version eFuse, the major version can be only <= 3. So it means that the old bootloader can detect correctly only chip version in range v0.0 - v3.0, where the minor version is always 0.

Please check the chip version using `esptool chip_id` command.

**API Reference**

**Header File**

- `components/hal/include/hal/efuse_hal.h`

**Functions**

- `void efuse_hal_get_mac(uint8_t *mac)`
  - Get factory mac address
- `uint32_t efuse_hal_chip_revision(void)`
  - Returns chip version.
  
  返回 Chip version in format: Major * 100 + Minor
- `uint32_t efuse_hal_get_major_chip_version(void)`
  - Returns major chip version.
- `uint32_t efuse_hal_get_minor_chip_version(void)`
  - Returns minor chip version.

### 2.10.5 控制台终端

ESP-IDF 提供了 console 组件，它包含了开发基于串口的交互式控制终端所需要的所有模块，主要支持以下功能：

- 由 linenoise 库具体实现，它支持处理退格键和方向键，支持回看命令的历史记录，支持命令的自动补全和参数提示。
- 将命令行拆分为参数列表。
- 参数解析，由 argtable3 库具体实现，该库提供解析 GNU 样式的命令行参数的 API。
- 用于注册和调度命令的函数。
- 帮助创建 REPL (Read-Evaluate-Print-Loop) 环境的函数。

**备注**：这些功能模块可以一起使用也可以独立使用，例如仅使用行编辑和命令注册的功能，然后使用 getopt 函数或者自定义的函数来实现参数解析，而不是直接使用 argtable3 库。同样地，还可以使用更简单的命令输入方法（比如 fgets 函数）和其他用于命令分割和参数解析的方法。
行编辑

行编辑功能允许用户通过按键插入编辑命令，使用退格键删除符号，使用左右键在命令中移动光标，使用上下键导航到之前输入的命令，使用制表键（“Tab”）来自动补全命令。

备注：此功能依赖于终端应用程序对 ANSI 转义符的支持。因此，显示原始 UART 数据的串口监视器不能与行编辑器一同使用。如果运行 system/console 示例程序时看到的输出结果是 [6n 或者类似的转义字符而不是命令行提示符 esp>` 时，就表明当前的串口监视器不支持 ANSI 转义字符。已知可用的串口监视器有 GNU screen、minicom 和 idf_monitor.py（可以通过在项目目录下执行`idf.py monitor`来调用）。

前往这里可以查看 linenoise 库提供的所有函数的描述。

配置 linenoise 库不需要显式地初始化，但是在调用行编辑函数之前，可能需要对某些配置的默认值稍作修改。

linenoiseClearScreen()

使用转义字符清除终端屏幕，并将光标定位在左上角。

linenoiseSetMultiLine()

在单行和多行编辑模式之间进行切换。单行模式下，如果命令的长度超过终端的宽度，会在行内滚动命令文本以显示文本的结尾。在这种情况下，文本的开头部分会被隐藏。单行模式在每次按下按键时发送给屏幕刷新的数据比较少，与多行模式相比更不容易发生故障。另一方面，在单行模式下编辑命令和复制命令将变得更加困难。默认情况下开启的是单行模式。

linenoiseAllowEmpty()

设置 linenoise 库收到空行的解析行为，设置为 true 时返回长度为零的字符串（""），设置为 false 时返回 NULL。默认情况下，将返回长度为零的字符串。

linenoiseSetMaxLineLen()

设置 linenoise 库中每行的最大长度。默认长度为 4096。如果需要优化 RAM 内存的使用，则可以通过这个函数设置一个小于默认 4 KB 的值来实现。

主循环 linenoise()

在大多数情况下，控制台应用程序都会具有相同的工作形式——在某个循环中不断读取输入的内容，然后解析再处理。linenoise() 是专门用来获取用户按键输入的函数。当回车键被按下后会返回完整的行内容。因此可以用它来完成前面循环中的“读取”任务。

linenoiseFree()

必须调用此函数才能释放从 linenoise() 函数获取的命令行缓冲区。

提示和补全 linenoiseSetCompletionCallback()

当用户按下制表键时，linenoise 会调用 补全回调函数，该回调函数会检查当前已经输入的内容，然后调用 linenoiseAddCompletion() 函数来提供所有可能的补全后的命令列表。启用补全功能，需要事先调用 linenoiseSetCompletionCallback() 函数来注册补全回调函数。

console 组件提供了一个现成的函数来为注册的命令提供补全功能esp_console_get_completion()（见下文）。

linenoiseAddCompletion()

补全回调函数会通过调用此函数来通知 linenoise 库当前键入命令所有可能的补全结果。

linenoiseSetHintsCallback()
每当用户的输入改变时，linenoise 就会调用此回调函数，检查到目前为止输入的命令行内容，然后提供带有提示信息的字符串（例如命令参数列表），然后会在同一行上用不同的颜色显示出该文本。

```
linenoiseSetFreeHintsCallback()
```

如果提示回调函数返回的提示字符串是动态分配的或者需要以其它方式回收，就需要使用 `linenoiseSetFreeHintsCallback()` 注册具体的清理函数。

### 历史记录

```
linenoiseHistorySetMaxLen()
```

该函数设置要保留在内存中的最近输入的命令的数量。用户通过使用向上/向下箭头来导航历史记录。

```
linenoiseHistoryAdd()
```

Linenoise 不会自动向历史记录中添加命令，应用程序需要调用此函数来将命令字符串添加到历史记录中。

```
linenoiseHistorySave()
```

该函数将命令的历史记录从 RAM 中保存为文本文件，例如保存到 SD 卡或者 Flash 的文件系统中。

```
linenoiseHistoryLoad()
```

与 `linenoiseHistorySave` 相对应，从文件中加载历史记录。

```
linenoiseHistoryFree()
```

释放用于存储命令历史记录的内存。当使用完 linenoise 库后需要调用此函数。

### 将命令行拆分成参数列表

console 组件提供 `esp_console_split_argv()` 函数来将命令行字符串拆分为参数列表。该函数会返回参数的数量 (argc) 和一个指针数组。该指针数组可以作为 argv 参数传递给任何接受 argc, argv 格式参数的函数。

根据以下规则来将命令行拆分成参数列表：

- 参数由空格分隔
- 如果参数本身需要使用空格，可以使用 `\`（反斜杠）对它们进行转义
- 其它能被识别的转义字符有 `\`（显示反斜杠本身）和 `\"`（显示双引号）
- 可以使用双引号来引用参数，引号只可能出现在参数的开头和结尾，参数中的引号必须如上所述进行转义。参数周围的引号会被 `esp_console_split_argv()` 函数删除

示例：

```
  abc def 1 20 .3 → [abc, def, 1, 20, .3]
  abc "123 456" def → [abc, 123 456, def]
  `a\ b\c" → [a b\c]
```

### 参数解析

对于参数解析，console 组件使用 argtable3 库。有关 argtable3 的介绍请查看 教程 或者 Github 仓库中的示例代码。

### 命令的注册与调度

console 组件包含了一些工具函数，用来注册命令，将用户输入的命令和已经注册的命令进行匹配，使用命令行输入的参数调用命令。
应用 程 序 首 先 调 用 `esp_console_init()` 来 初始化 命 令 注 册 模 块，然 后 调 用 `esp_console_cmd_register()` 函数注册命令处理程序。

对于每个命令，应用程序需要提供以下信息（需要以 `esp_console_cmd_t` 结构体的形式给出）：

- 命令名字（不含空格的字符串）
- 帮助文档，解释该命令的用途
- 可选的提示文本，列出命令的参数。如果应用程序使用 `Argtable3` 库来解析参数，则可以通过提供指向 `Argtable` 参数定义结构体的指针来自动生成提示文本
- 命令处理函数

命令注册模块还提供了其它函数：

`esp_console_run()`

该函数接受命令行字符串，使用 `esp_console_split_argv()` 函数将其拆分为 `argv` 形式的参数列表，在已经注册的组件列表中查找命令，如果找到，则执行其对应的处理程序。

`esp_console_register_help_command()`

将 `help` 命令添加到已注册命令列表中，此命令将会以列表的方式打印所有注册的命令及其参数和帮助文本。

`esp_console_get_completion()`

与 `linenoise` 库中的 `linenoiseSetCompletionCallback()` 一同使用的回调函数，根据已经注册的命令列表为 `linenoise` 提供补全功能。

`esp_console_get_hint()`

与 `linenoise` 库中的 `linenoiseSetHintsCallback()` 一同使用的回调函数，为 `linenoise` 提供已经注册的命令的参数提示功能。

初始化 REPL 环境

除了上述的各种函数，`console` 组件还提供了一些 API 来帮助创建一个基本的 REPL 环境。

在一个典型的 `console` 应用中，你只需要调用 `esp_console_new_repl_uart()`，它会为你初始化好构建在 UART 基础上的 REPL 环境，其中包括安装 UART 驱动，基本的 `console` 配置，创建一个新的线程来执行 REPL 任务，注册一些基本的命令（比如 `help` 命令）。

之后你可以使用 `esp_console_cmd_register()` 来注册其它命令。REPL 环境在初始化后需要再调用 `esp_console_start_repl()` 函数才能开始运行。

应用程序示例

`system/console` 目录下提供了 `console` 组件的示例应用程序，展示了具体的使用方法。该示例介绍了如何初始化 UART 和 VFS 的功能，设置 `linenoise` 库，从 UART 中读取命令并加以处理，然后将历史命令存储到 Flash 中。更多信息，请参阅示例代码目录中的 README.md 文件。

此外，ESP-IDF 还提供了众多基于 `console` 组件的示例程序，它们可以辅助应用程序的开发。例如，`peripherals/i2c/i2c_tools`, `wifi/iperf` 等等。

API 参考

Header File

- components/console/esp_console.h
Chapter 2. API reference

**Functions**

`esp_err_t esp_console_init(const esp_console_config_t *config)`
initialize console module

**备注:** Call this once before using other console module features

**参数** config – console configuration

**返回**
- ESP_OK on success
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_STATE if already initialized
- ESP_ERR_INVALID_ARG if the configuration is invalid

`esp_err_t esp_console_deinit(void)`
de-initialize console module

**备注:** Call this once when done using console module functions

**返回**
- ESP_OK on success
- ESP_ERR_INVALID_STATE if not initialized yet

`esp_err_t esp_console_cmd_register(const esp_console_cmd_t *cmd)`
Register console command.

**参数** cmd – pointer to the command description; can point to a temporary value

**返回**
- ESP_OK on success
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_ARG if command description includes invalid arguments

`esp_err_t esp_console_run(const char *cmdline, int *cmd_ret)`
Run command line.

**参数**
- cmdline – command line (command name followed by a number of arguments)
- cmd_ret – [out] return code from the command (set if command was run)

**返回**
- ESP_OK, if command was run
- ESP_ERR_INVALID_ARG, if the command line is empty, or only contained whitespace
- ESP_ERR_NOTFOUND, if command with given name wasn’t registered
- ESP_ERR_INVALID_STATE, if esp_console_init wasn’t called

`size_t esp_console_split_argv(char *line, char **argv, size_t argv_size)`
Split command line into arguments in place.

* - This function finds whitespace-separated arguments in the given input line.
* - 'abc def 1 20 .3' -> ['abc', 'def', '1', '20', '.3']
* - Argument which include spaces may be surrounded with quotes. In this case spaces are preserved and quotes are stripped.
* - 'abc "123 456" def' -> ['abc', '123 456', 'def']
Chapter 2. API 参考

```
* - Escape sequences may be used to produce backslash, double quote, and space:
  * 'a\ b\c"' -> [ 'a b\c"' ]
```

备注: Pointers to at most argv_size - 1 arguments are returned in argv array. The pointer after the last one (i.e. argv[argc]) is set to NULL.

参数
- line - pointer to buffer to parse; it is modified in place
- argv - array where the pointers to arguments are written
- argv_size - number of elements in argv_array (max. number of arguments)
- argc - number of arguments found

返回 number of arguments found (argc)

```c
void esp_console_get_completion(const char* buf, linenoiseCompletions *lc)
```

Callback which provides command completion for linenoise library.

When using linenoise for line editing, command completion support can be enabled like this:

```
linenoiseSetCompletionCallback(&esp_console_get_completion);
```

参数
- buf - the string typed by the user
- lc - linenoiseCompletions to be filled in

```c
const char *esp_console_get_hint(const char* buf, int*color, int*bold)
```

Callback which provides command hints for linenoise library.

When using linenoise for line editing, hints support can be enabled as follows:

```
linenoiseSetHintsCallback((linenoiseHintsCallback*)&esp_console_get_hint);
```

The extra cast is needed because linenoiseHintsCallback is defined as returning a char* instead of const char*.

参数
- buf - line typed by the user
- color - [out] ANSI color code to be used when displaying the hint
- bold - [out] set to 1 if hint has to be displayed in bold

返回 string containing the hint text. This string is persistent and should not be freed (i.e. linenoiseSetFreeHintsCallback should not be used).

```c
esp_err_t esp_console_register_help_command(void)
```

Register a ‘help’ command.

Default ‘help’ command prints the list of registered commands along with hints and help strings.

返回
- ESP_OK on success
- ESP_ERR_INVALID_STATE, if esp_console_init wasn’t called

```c
esp_err_t esp_console_new_repl_uart(const esp_console_dev_uart_config_t *dev_config, const esp_console_repl_config_t *repl_config, esp_console_repl_t **ret_repl)
```

Establish a console REPL environment over UART driver.

**Attention** This function is meant to be used in the examples to make the code more compact. Applications which use console functionality should be based on the underlyinglinenoise and esp_console functions.

备注: This is an all-in-one function to establish the environment needed for REPL, includes:
• Install the UART driver on the console UART (8n1, 115200, REF_TICK clock source)
• Configures the stdin/stdout to go through the UART driver
• Initializes linenoise
• Spawn new thread to run REPL in the background

### Parameters
- **dev_config** - [in] UART device configuration
- **repl_config** - [in] REPL configuration
- **ret_repl** - [out] return REPL handle after initialization succeed, return NULL otherwise

### Return
- ESP_OK on success
- ESP_FAIL Parameter error

```c
esp_err_t esp_console_new_repl_usb_cdc(const esp_console_dev_usb_cdc_config_t *dev_config, const esp_console_repl_config_t *repl_config, esp_console_repl_t **ret_repl)
```

Establish a console REPL environment over USB CDC.

**Attention** This function is meant to be used in the examples to make the code more compact. Applications which use console functionality should be based on the underlying linenoise and esp_console functions.

### Structures
- **struct esp_console_config_t**
  Parameters for console initialization.
Public Members

size_t max_cmdline_length
    length of command line buffer, in bytes

size_t max_cmdline_args
    maximum number of command line arguments to parse

int hint_color
    ASCII color code of hint text.

int hint_bold
    Set to 1 to print hint text in bold.

struct esp_console_repl_config_t
    Parameters for console REPL (Read Eval Print Loop)

Public Members

uint32_t max_history_len
    maximum length for the history

const char *history_save_path
    file path used to save history commands, set to NULL won’t save to filesystem

uint32_t task_stack_size
    repl task stack size

uint32_t task_priority
    repl task priority

const char *prompt
    prompt (NULL represents default: “esp> “)

size_t max_cmdline_length
    maximum length of a command line. If 0, default value will be used

struct esp_console_dev_uart_config_t
    Parameters for console device: UART.

Public Members

int channel
    UART channel number (count from zero)

int baud_rate
    Communication baud rate.
int `tx_gpio_num`
   GPIO number for TX path, -1 means using default one.

int `rx_gpio_num`
   GPIO number for RX path, -1 means using default one.

struct `esp_console_dev_usb_cdc_config_t`
   Parameters for console device: USB CDC.

---

备注: It’s an empty structure for now, reserved for future

---

struct `esp_console_cmd_t`
   Console command description.

   **Public Members**

   const char *`command`
      Command name. Must not be NULL, must not contain spaces. The pointer must be valid until the call to `esp_console_deinit`.

   const char *`help`
      Help text for the command, shown by help command. If set, the pointer must be valid until the call to `esp_console_deinit`. If not set, the command will not be listed in ‘help’ output.

   const char *`hint`
      Hint text, usually lists possible arguments. If set to NULL, and ‘argtable’ field is non-NULL, hint will be generated automatically

   `esp_console_cmd_func_t` `func`
      Pointer to a function which implements the command.

   void *`argtable`
      Array or structure of pointersto `arg_xxx` structures, may be NULL. Used to generate hint text if ‘hint’ is set to NULL. Array/structure which this field points to must end with an arg_end. Only used for the duration of `esp_console_cmd_register` call.

struct `esp_console_repl_s`
   Console REPL base structure.

   **Public Members**

   `esp_err_t (*del)(esp_console_repl_t *repl)`
      Delete console REPL environment.
      
      **Param** repl [in] REPL handle returned from `esp_console_new_repl_xxx`
      **Return**
      - ESP_OK on success
      - ESP_FAIL on errors
Chapter 2. API

Macros

ESP_CONSOLE_CONFIG_DEFAULT()
Default console configuration value.

ESP_CONSOLE_REPL_CONFIG_DEFAULT()
Default console repl configuration value.

ESP_CONSOLE_DEV_UART_CONFIG_DEFAULT()

ESP_CONSOLE_DEV_CDC_CONFIG_DEFAULT()

Type Definitions

typedef struct linenoiseCompletions linenoiseCompletions
typedef int (*esp_console_cmd_func_t)(int argc, char **argv)
Console command main function.

Param argc number of arguments
Param argv array with argc entries, each pointing to a zero-terminated string argument
Return console command return code, 0 indicates "success"

typedef struct esp_console_repl_s esp_console_repl_t
Type defined for console REPL.

2.10.6 eFuse Manager

Introduction

The eFuse Manager library is designed to structure access to eFuse bits and make using these easy. This library operates eFuse bits by a structure name which is assigned in eFuse table. This sections introduces some concepts used by eFuse Manager.

Hardware description

The ESP32 has a number of eFuses which can store system and user parameters. Each eFuse is a one-bit field which can be programmed to 1 after which it cannot be reverted back to 0. Some of system parameters are using these eFuse bits directly by hardware modules and have special place (for example EFUSE_BLK0).

For more details, see ESP32 Technical Reference Manual > eFuse Controller (eFuse) [PDF]. Some eFuse bits are available for user applications.

ESP32 has 4 eFuse blocks each of the size of 256 bits (not all bits are available):

- EFUSE_BLK0 is used entirely for system purposes;
- EFUSE_BLK1 is used for flash encrypt key. If not using that Flash Encryption feature, they can be used for another purpose;
- EFUSE_BLK2 is used for security boot key. If not using that Secure Boot feature, they can be used for another purpose;
- EFUSE_BLK3 can be partially reserved for the custom MAC address, or used entirely for user application. Note that some bits are already used in IDF.

Each block is divided into 8 32-bits registers.
eFuse Manager component

The component has API functions for reading and writing fields. Access to the fields is carried out through the structures that describe the location of the eFuse bits in the blocks. The component provides the ability to form fields of any length and from any number of individual bits. The description of the fields is made in a CSV file in a table form. To generate from a tabular form (CSV file) in the C-source uses the tool efuse_table_gen.py. The tool checks the CSV file for uniqueness of field names and bit intersection, in case of using a custom file from the user’s project directory, the utility will check with the common CSV file.

CSV files:

- common (esp Efuse_table.csv) - contains eFuse fields which are used inside the IDF. C-source generation should be done manually when changing this file (run command idf.py efuse-common-table). Note that changes in this file can lead to incorrect operation.
- custom - (optional and can be enabled by CONFIG_EFUSE_CUSTOM_TABLE) contains eFuse fields that are used by the user in their application. C-source generation should be done manually when changing this file and running idf.py efuse-custom-table.

Description CSV file

The CSV file contains a description of the eFuse fields. In the simple case, one field has one line of description. Table header:

| # field_name, efuse_block(EFUSE_BLK0..EFUSE_BLK3), bit_start(0..255), bit_count(1..256), comment |

Individual params in CSV file the following meanings:

- **field_name** Name of field. The prefix ESP_EFUSE_ will be added to the name, and this field name will be available in the code. This name will be used to access the fields. The name must be unique for all fields. If the line has an empty name, then this line is combined with the previous field. This allows you to set an arbitrary order of bits in the field, and expand the field as well (see MAC_FACTORY field in the common table). The field name supports structured format using . to show that the field belongs to another field (see WR_DIS and RD_DIS in the common table).
- **efuse_block** Block number. It determines where the eFuse bits will be placed for this field. Available EFUSE_BLK0..EFUSE_BLK3.
- **bit_start** Start bit number (0..255). The bit_start field can be omitted. In this case, it will be set to bit_start + bit_count from the previous record, if it has the same efuse_block. Otherwise (if efuse_block is different, or this is the first entry), an error will be generated.
- **bit_count** The number of bits to use in this field (1..-). This parameter can not be omitted. This field also may be MAX_BLK_LEN in this case, the field length will have the maximum block length, taking into account the coding scheme (applicable for ESP_EFUSE_SECURE_BOOT_KEY and ESP_EFUSE_ENCRYPT_FLASH_KEY fields). The value MAX_BLK_LEN depends on CONFIG_EFUSE_CODE_SCHEME_SELECTOR, will be replaced with “None” - 256, “3/4” - 192, “REPEAT” - 128.
- **comment** This param is using for comment field, it also move to C-header file. The comment field can be omitted.

If a non-sequential bit order is required to describe a field, then the field description in the following lines should be continued without specifying a name, this will indicate that it belongs to one field. For example two fields MAC_FACTORY and MAC_FACTORY_CRC:

<table>
<thead>
<tr>
<th># Factory MAC address #</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC_FACTORY, EFUSE_BLK0, 72, 8, Factory MAC addr [0]</td>
</tr>
<tr>
<td>EFUSE_BLK0, 64, 8, Factory MAC addr [1]</td>
</tr>
<tr>
<td>EFUSE_BLK0, 56, 8, Factory MAC addr [2]</td>
</tr>
<tr>
<td>EFUSE_BLK0, 48, 8, Factory MAC addr [3]</td>
</tr>
<tr>
<td>EFUSE_BLK0, 40, 8, Factory MAC addr [4]</td>
</tr>
</tbody>
</table>
This field will available in code as ESP_EFUSE_MAC_FACTORY and ESP_EFUSE_MAC_FACTORY_CRC.

**Structured efuse fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Block</th>
<th>Bit Start</th>
<th>Bit End</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR_DIS</td>
<td>EFUSE_BLK0</td>
<td>0</td>
<td>32</td>
<td>Write protection</td>
</tr>
<tr>
<td>WR_DIS.RD_DIS</td>
<td>EFUSE_BLK0</td>
<td>0</td>
<td>1</td>
<td>Write protection for..</td>
</tr>
<tr>
<td>WR_DIS.FIELD_1</td>
<td>EFUSE_BLK0</td>
<td>1</td>
<td>1</td>
<td>Write protection for..</td>
</tr>
<tr>
<td>WR_DIS.FIELD_2</td>
<td>EFUSE_BLK0</td>
<td>2</td>
<td>2</td>
<td>Write protection for..</td>
</tr>
<tr>
<td>WR_DIS.FIELD_2.B1</td>
<td>EFUSE_BLK0</td>
<td>4</td>
<td>2</td>
<td>Write protection for..</td>
</tr>
<tr>
<td>WR_DIS.FIELD_2.B2</td>
<td>EFUSE_BLK0</td>
<td>5</td>
<td>1</td>
<td>Write protection for..</td>
</tr>
<tr>
<td>WR_DIS.FIELD_3</td>
<td>EFUSE_BLK0</td>
<td>7</td>
<td>1</td>
<td>Write protection for..</td>
</tr>
</tbody>
</table>

The structured eFuse field looks like WR_DIS.RD_DIS where the dot points that this field belongs to the parent field - WR_DIS and can not be out of the parent’s range.

It is possible to use some levels of structured fields as WR_DIS.FIELD_2.B1 and B2. These fields should not be crossed each other and should be in the range of two fields: WR_DIS and WR_DIS.FIELD_2.

It is possible to create aliases for fields with the same range, see WR_DIS.FIELD_3 and WR_DIS.FIELD_3.ALIAS.

The IDF names for structured efuse fields should be unique. The efuse_table_gen tool will generate the final names where the dot will be replaced by _.. The names for using in IDF are ESP_EFUSE_WR_DIS, ESP_EFUSE_WR_DIS.RD_DIS, ESP_EFUSE_WR_DIS.FIELD_2.B1, etc.

The efuse_table_gen tool checks that the fields do not overlap each other and must be within the range of a field if there is a violation, then throws the following error:

**Solution:** Describe SERIAL_NUMBER to be included in USER_DATA.(USER_DATA.SERIAL_NUMBER).

**Solution:** Change bit_start for FIELD.MAJOR_NUMBER from 60 to 0, so MAJOR_NUMBER is in the FIELD range.

**efuse_table_gen.py tool**

The tool is designed to generate C-source files from CSV file and validate fields. First of all, the check is carried out on the uniqueness of the names and overlaps of the field bits. If an additional custom file is used, it will be checked with the existing common file (esp_efuse_table.csv). In case of errors, a message will be displayed and the string that caused the error. C-source files contain structures of type esp_efuse_desc_t.

To generate a common files, use the following command idf.py efuse-common-table or:
After generation in the folder `$IDF_PATH/components/efuse/esp32` create:

- `esp_efuse_table.c` file.
- In include folder `esp_efuse_table.c` file.

To generate a custom files, use the following command `idf.py efuse-custom-table` or:

```
cd $IDF_PATH/components/efuse/
./efuse_table_gen.py --idf_target esp32 esp32/esp_efuse_table.csv
```

After generation in the folder `PROJECT_PATH/main` create:

- `esp_efuse_custom_table.c` file.
- In include folder `esp_efuse_custom_table.c` file.

To use the generated fields, you need to include two files:

```
#include "esp_efuse.h"
#include "esp_efuse_table.h" // or "esp_efuse_custom_table.h"
```

**Supported coding scheme**

eFuse have three coding schemes:

- None (value 0).
- 3/4 (value 1).
- Repeat (value 2).

The coding scheme affects only EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3 blocks. EUSE_BLK0 block always has a coding scheme None. Coding changes the number of bits that can be written into a block, the block length is constant 256, some of these bits are used for encoding and not available for the user.

When using a coding scheme, the length of the payload that can be written is limited (for more details 20.3.1.3 System Parameter coding_scheme):

- None 256 bits.
- 3/4 192 bits.
- Repeat 128 bits.

You can find out the coding scheme of your chip:

- run a `espefuse.py -p PORT summary` command.
- from `esptool` utility logs (during flashing).
- calling the function in the code `esp_efuse_get_coding_scheme()` for the EFUSE_BLK3 block.

eFuse tables must always comply with the coding scheme in the chip. There is an `CONFIG_EFUSE_CODE_SCHEME_SELECTOR` option to select the coding type for tables in a Kconfig. When generating source files, if your tables do not follow the coding scheme, an error message will be displayed. Adjust the length or offset fields. If your program was compiled with None encoding and 3/4 is used in the chip, then the `ESP_ERR_CODING` error may occur when calling the eFuse API (the field is outside the block boundaries). If the field matches the new block boundaries, then the API will work without errors.

Also, 3/4 coding scheme imposes restrictions on writing bits belonging to one coding unit. The whole block with a length of 256 bits is divided into 4 coding units, and in each coding unit there are 6 bytes of useful data and 2 service bytes. These 2 service bytes contain the checksum of the previous 6 data bytes.

It turns out that only one field can be written into one coding unit. Repeated rewriting in one coding unit is prohibited. But if the record was made in advance or through a `esp_efuse_write_block()` function, then reading the fields belonging to one coding unit is possible.
In case 3/4 coding scheme, the writing process is divided into the coding units and we cannot use the usual mode of writing some fields. We can prepare all the data for writing and burn it in one time. You can also use this mode for None coding scheme but it is not necessary. It is important for 3/4 coding scheme. The batch writing mode blocks esp_efuse_read... operations.

After changing the coding scheme, run efuse_common_table and efuse_custom_table commands to check the tables of the new coding scheme.

To write some fields into one block, or different blocks in one time, you need to use the batch writing mode. Firstly set this mode through esp_efuse_batch_write_begin() function then write some fields as usual using the esp_efuse_write... functions. At the end to burn them, call the esp_efuse_batch_write_commit() function. It burns prepared data to the eFuse blocks and disables the batch recording mode.

备注: If there is already pre-written data in the eFuse block using the 3/4 or Repeat encoding scheme, then it is not possible to write anything extra (even if the required bits are empty) without breaking the previous encoding data. This encoding data will be overwritten with new encoding data and completely destroyed (however, the payload eFuses are not damaged). It can be related to: CUSTOM_MAC, SPI_PAD_CONFIG_HD, SPI_PAD_CONFIG_CS, etc. Please contact Espressif to order the required pre-burnt eFuses.

FOR TESTING ONLY (NOT RECOMMENDED): You can ignore or suppress errors that violate encoding scheme data in order to burn the necessary bits in the eFuse block.

eFuse API

Access to the fields is via a pointer to the description structure. API functions have some basic operation:

- esp_efuse_read_field_blob() - returns an array of read eFuse bits.
- esp_efuse_read_field_cnt() - returns the number of bits programmed as “1”.
- esp_efuse_write_field_blob() - writes an array.
- esp_efuse_write_field_cnt() - writes a required count of bits as “1”.
- esp_efuse_get_field_size() - returns the number of bits by the field name.
- esp_efuse_read_reg() - returns value of eFuse register.
- esp_efuse_write_reg() - writes value to eFuse register.
- esp_efuse_get_coding_scheme() - returns eFuse coding scheme for blocks.
- esp_efuse_read_block() - reads key to eFuse block starting at the offset and the required size.
- esp_efuse_write_block() - writes key to eFuse block starting at the offset and the required size.
- esp_efuse_batch_write_begin() - set the batch mode of writing fields.
- esp_efuse_batch_write_commit() - writes all prepared data for batch writing mode and reset the batch writing mode.
- esp_efuse_batch_write_cancel() - reset the batch writing mode and prepared data.
- esp_efuse_get_key_dis_read() - Returns a read protection for the key block.
- esp_efuse_set_key_dis_read() - Sets a read protection for the key block.
- esp_efuse_get_key_dis_write() - Returns a write protection for the key block.
- esp_efuse_set_key_dis_write() - Sets a write protection for the key block.
- esp_efuse_get_key_purpose() - Returns the current purpose set for an eFuse key block.
- esp_efuse_write_key() - Programs a block of key data to an eFuse block
- esp_efuse_write_keys() - Programs keys to unused eFuse blocks
- esp_efuse_find_purpose() - Finds a key block with the particular purpose set.
- esp_efuse_get_keypurpose_dis_write() - Returns a write protection of the key purpose field for an eFuse key block (for esp32 always true).
- esp_efuse_key_block_unused() - Returns true if the key block is unused, false otherwise.

For frequently used fields, special functions are made, like this esp_efuse_get_pkg_ver().
## How to add a new field

1. Find a free bits for field. Show `esp_efuse_table.csv` file or run `idf.py show-efuse-table` or the next command:

```
$ ./efuse_table_gen.py esp32/esp_efuse_table.csv --info
```

Parsing efuse CSV input file $IDF_PATH/components/efuse/esp32/esp_efuse_table.csv...

Verifying efuse table...

Max number of bits in BLK 192

<table>
<thead>
<tr>
<th>Sorted efuse table:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>field_name</td>
<td>efuse_block</td>
</tr>
<tr>
<td>1</td>
<td>WR_DIS_EFUSE_RD_DISABLE</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>2</td>
<td>WR_DIS_FLASH_CRYPT_CNT</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>3</td>
<td>WR_DIS_BLK1</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>4</td>
<td>WR_DIS_BLK2</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>5</td>
<td>WR_DIS_BLK3</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>6</td>
<td>RD_DIS_BLK1</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>7</td>
<td>RD_DIS_BLK2</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>8</td>
<td>RD_DIS_BLK3</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>9</td>
<td>FLASH_CRYPT_CNT</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>10</td>
<td>UART_DOWNLOAD_DIS</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>11</td>
<td>MAC_FACTORY</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>12</td>
<td>MAC_FACTORY</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>13</td>
<td>MAC_FACTORY</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>14</td>
<td>MAC_FACTORY</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>15</td>
<td>MAC_FACTORY</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>16</td>
<td>MAC_FACTORY</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>17</td>
<td>MAC_FACTORY_CRC</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>18</td>
<td>CHIP_VER_DIS_APP_CPU</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>19</td>
<td>CHIP_VER_DIS_BT</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>20</td>
<td>CHIP_VER_PKG</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>21</td>
<td>CHIP_CPU_FREQ_LOW</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>22</td>
<td>CHIP_CPU_FREQ_BT</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>23</td>
<td>CHIP_CPU_FREQ_RATED</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>24</td>
<td>CHIP_VER_REV1</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>25</td>
<td>ADC_VREF_AND_SDIO_DREF</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>26</td>
<td>XPD_SDIO_REG</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>27</td>
<td>SDIO_TIEH</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>28</td>
<td>SDIO_FORCE</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>29</td>
<td>CHIP_VER_REV2</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>30</td>
<td>ENCRYPT_CONFIG</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>31</td>
<td>CONSOLE_DEBUG_DISABLE</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>32</td>
<td>ABS_DONE_0</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>33</td>
<td>ABS_DONE_1</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>34</td>
<td>DISABLE_JTAG</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>35</td>
<td>DISABLE_DL_ENCRYPT</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>36</td>
<td>DISABLE_DL_DECRYPT</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>37</td>
<td>DISABLE_DL_CACHE</td>
<td>EFUSE_BLK0</td>
</tr>
<tr>
<td>38</td>
<td>ENCRYPT_FLASH_KEY</td>
<td>EFUSE_BLK1</td>
</tr>
<tr>
<td>39</td>
<td>SECURE_BOOT_KEY</td>
<td>EFUSE_BLK2</td>
</tr>
<tr>
<td>40</td>
<td>MAC_CUSTOM_CRC</td>
<td>EFUSE_BLK3</td>
</tr>
<tr>
<td>41</td>
<td>MAC_CUSTOM</td>
<td>EFUSE_BLK3</td>
</tr>
<tr>
<td>42</td>
<td>ADC1_TP_LOW</td>
<td>EFUSE_BLK3</td>
</tr>
<tr>
<td>43</td>
<td>ADC1_TP_HIGH</td>
<td>EFUSE_BLK3</td>
</tr>
<tr>
<td>44</td>
<td>ADC2_TP_LOW</td>
<td>EFUSE_BLK3</td>
</tr>
<tr>
<td>45</td>
<td>ADC2_TP_HIGH</td>
<td>EFUSE_BLK3</td>
</tr>
<tr>
<td>46</td>
<td>SECURE_VERSION</td>
<td>EFUSE_BLK3</td>
</tr>
<tr>
<td>47</td>
<td>MAC_CUSTOM_VER</td>
<td>EFUSE_BLK3</td>
</tr>
</tbody>
</table>

Used bits in efuse table:
The number of bits not included in square brackets is free (some bits are reserved for Espressif). All fields are checked for overlapping.

To add fields to an existing field, use the *Structured efuse fields* technique. For example, adding the fields: SERIAL_NUMBER, MODEL_NUMBER and HARDWARE REV to an existing USER_DATA field. Use . (dot) to show an attachment in a field.

2. Fill a line for field: field_name, efuse_block, bit_start, bit_count, comment.

3. Run a `show_efuse_table` command to check eFuse table. To generate source files run `efuse_common_table` or `efuse_custom_table` command.

You may get errors such as intersects with or out of range. Please see how to solve them in the *Structured efuse fields* article.

**Bit Order**

The eFuses bit order is little endian (see the example below), it means that eFuse bits are read and written from LSB to MSB:

```
$ espefuse.py dump
```

```none
USER_DATA (BLOCK3) [3 ] read_regs: 03020100 07060504 0B0A0908
<RF0E0D00C 13121111 17161514 1B1A1918 1F1E1D1C
BLOCK4 (BLOCK4) [4 ] read_regs: 03020100 07060504 0B0A0908
<RF0E0D00C 13121111 17161514 1B1A1918 1F1E1D1C
```

where is the register representation:

```none
EFUSE_RD_USR_DATA0_REG = 0x03020100
EFUSE_RD_USR_DATA1_REG = 0x07060504
EFUSE_RD_USR_DATA2_REG = 0x0B0A0908
EFUSE_RD_USR_DATA3_REG = 0x0F0E0D0C
EFUSE_RD_USR_DATA4_REG = 0x13121111
EFUSE_RD_USR_DATA5_REG = 0x17161514
EFUSE_RD_USR_DATA6_REG = 0x1B1A1918
EFUSE_RD_USR_DATA7_REG = 0x1F1E1D1C
```

where is the byte representation:

```none
byte[0] = 0x00, byte[1] = 0x01, ... byte[3] = 0x03, byte[4] = 0x04, ..., byte[31] = 0x1F
```

(下页继续)
For example, the csv file describes the `USER_DATA` field, which occupies all 256 bits (a whole block).

```
| USER_DATA, | 0, 256, User data |
| USER_DATA.FIELD1, | 16, 16, Field1 |
| ID, | 8, 3, ID bit[0..2] |
| | 16, 2, ID bit[3..4] |
| | 32, 3, ID bit[5..7] |
```

Thus, reading the eFuse USER_DATA block written as above gives the following results:

```
uint8_t buf[32] = { 0 };
esp_efuse_read_field_blob(ESP_EFUSE_USER_DATA, &buf, sizeof(buf) * 8);
// buf[0] = 0x00, buf[1] = 0x01, ... buf[31] = 0x1F

uint32_t field1 = 0;
size_t field1_size = ESP_EFUSE_USER_DATA[0]->bit_count;  // can be used for this case because it only consists of one entry
esp_efuse_read_field_blob(ESP_EFUSE_USER_DATA, &field1, field1_size);
// field1 = 0x0302

uint32_t field1_1 = 0;
esp_efuse_read_field_blob(ESP_EFUSE_USER_DATA, &field1_1, 2);  // reads only first 2 bits
// field1 = 0x0002

uint8_t id = 0;
size_t id_size = esp_efuse_get_field_size(ESP_EFUSE_ID);  // returns 6
// size_t id_size = ESP_EFUSE_USER_DATA[0]->bit_count;  // can NOT be used because it consists of 3 entries. It returns 3 not 6.
esp_efuse_read_field_blob(ESP_EFUSE_ID, &id, id_size);
// id = 0x91
// b'100 10 001
// [3] [2] [3]

uint8_t id_1 = 0;
esp_efuse_read_field_blob(ESP_EFUSE_ID, &id_1, 3);
// id = 0x01
// b'001
```

**Debug eFuse & Unit tests**

**Virtual eFuses**  The Kconfig option `CONFIG_EFUSE_VIRTUAL` will virtualize eFuse values inside the eFuse Manager, so writes are emulated and no eFuse values are permanently changed. This can be useful for debugging app and unit tests. During startup, the eFuses are copied to RAM. All eFuse operations (read and write) are performed with RAM instead of the real eFuse registers.

In addition to the `CONFIG_EFUSE_VIRTUAL` option there is `CONFIG_EFUSE_VIRTUAL_KEEP_IN_FLASH` option that adds a feature to keep eFuses in flash memory. To use this mode the partition_table should have the `efuse_em` partition. `partition.csv`: "efuse_em, data, efuse, , 0x2000,". During startup, the eFuses are copied from flash or, in case if flash is empty, from real eFuse to RAM and then update flash. This option allows keeping eFuses after reboots (possible to test secure _boot and flash_encryption features with this option).

**espefuse.py**  esptool includes a useful tool for reading/writing ESP32 eFuse bits - espefuse.py.

```
espefuse.py -p PORT summary
```
Connecting........__
Detecting chip type... ESP32
espefuse.py v3.1-dev

EFUSE_NAME (Block) Description = [Meaningful Value]__,
-> [Readable/Writeable] (Hex Value)
-----------------------------------------------------------------------------------------------------------------------------------

Calibration fuses:
BLK3_PART_RESERVE (BLOCK0): BLOCK3 partially served for ADC__
-> calibration data = True R/W (0b1)
ADC_VREF (BLOCK0): Voltage reference calibration __
-> 1114 R/W (0b00010)
ADC1_TP_LOW (BLOCK3): ADC1 150mV reading __
-> 346 R/W (0b0010001)
ADC1_TP_HIGH (BLOCK3): ADC1 850mV reading __
-> 3285 R/W (0b000000101)
ADC2_TP_LOW (BLOCK3): ADC2 150mV reading __
-> 449 R/W (0b00001111)
ADC2_TP_HIGH (BLOCK3): ADC2 850mV reading __
-> 3362 R/W (0b11110101)

Config fuses:
XPD_SDIO_FORCE (BLOCK0): Ignore MTDI pin (GPIO12) for VDD_SDIO on__
-> reset = False R/W (0b0)
XPD_SDIO_REG (BLOCK0): If XPD_SDIO_FORCE, enable VDD_SDIO reg on__
-> reset = False R/W (0b0)
XPD_SDIO_TIEH (BLOCK0): If XPD_SDIO_FORCE & XPD_SDIO_REG __
-> 1.8V R/W (0b0)
CLK8M_FREQ (BLOCK0): 8MHz clock freq override __
-> 53 R/W (0x35)
SPI_PAD_CONFIG_CLK (BLOCK0): Override SD_CLK pad (GPIO6/SPICLK) __
-> 0 R/W (0b000000)
SPI_PAD_CONFIG_Q (BLOCK0): Override SD_DATA_0 pad (GPIO7/SPIQ) __
-> 0 R/W (0b000000)
SPI_PAD_CONFIG_D (BLOCK0): Override SD_DATA_1 pad (GPIO8/SPID) __
-> 0 R/W (0b000000)
SPI_PAD_CONFIG_HD (BLOCK0): Override SD_DATA_2 pad (GPIO9/SPIHD) __
-> 0 R/W (0b000000)
SPI_PAD_CONFIG_CS0 (BLOCK0): Override SD_CMD pad (GPIO11/SPICS0) __
-> 0 R/W (0b000000)
DISABLE_SDIO_HOST (BLOCK0): Disable SDIO host __
-> False R/W (0b0)

Efuse fuses:
WR_DIS (BLOCK0): Efuse write disable mask __
-> 0 R/W (0x0000)
RD_DIS (BLOCK0): Efuse read disable mask __
-> 0 R/W (0x0)
CODING_SCHEME (BLOCK0): Efuse variable block length scheme __
= 3/4 (BLK1-3 len=192 bits) R/W (0b01)
KEY_STATUS (BLOCK0): Usage of efuse block 3 (reserved) __
-> False R/W (0b0)

Identity fuses:
MAC (BLOCK0): Factory MAC Address __
= 84:0d:8e:18:8e:44 (CRC 0xad OK) R/W
MAC_CRC (BLOCK0): CRC8 for factory MAC address __
= 173 R/W (0xad)
CHIP_VER_REV1 (BLOCK0): Silicon Revision 1 __
-> True R/W (0b1)
CHIP_VER_REV2 (BLOCK0): Silicon Revision 2 __
-> False R/W (0b0)
Chapter 2. API Reference

CHIP_VERSION (BLOCK0): Reserved for future chip versions
- = 2 R/W (0b10)
CHIP_PACKAGE (BLOCK0): Chip package identifier
- = 0 R/W (0b000)
MAC_VERSION (BLOCK3): Version of the MAC field
- = 0 R/W (0x00)

Security fuses:
FLASH_CRYPT_CNT (BLOCK0): Flash encryption mode counter
- = 0 R/W (0b0000000)
UART_DOWNLOAD_DIS (BLOCK0): Disable UART download mode (ESP32 rev3)
-only = False R/W (0b0)
FLASH_CRYPT_CONFIG (BLOCK0): Flash encryption config (key tweak bits)
- = 0 R/W (0x0)
CONSOLE_DEBUG_DISABLE (BLOCK0): Disable ROM BASIC interpreter fallback
- = True R/W (0b1)
ABS_DONE_0 (BLOCK0):
- image = False R/W (0b0)
ABS_DONE_1 (BLOCK0):
- image = False R/W (0b0)
JTAG_DISABLE (BLOCK0):
- = False R/W (0b0)
DISABLE_DL_ENC (BLOCK0): Disable flash encryption in UART
- bootloader = False R/W (0b0)
DISABLE_DL_DEC (BLOCK0): Disable flash decryption in UART
- bootloader = False R/W (0b0)
DISABLE_DL_CACHE (BLOCK0): Disable flash cache in UART bootloader
- = False R/W (0b0)
BLOCK1 (BLOCK1): Flash encryption key
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
BLOCK2 (BLOCK2): Secure boot key
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
BLOCK3 (BLOCK3): Variable Block 3
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W

Flash voltage (VDD_SDIO) determined by GPIO12 on reset (High for 1.8V, Low/NC for
- 3.3V).

To get a dump for all eFuse registers.

```
esefuse.py -p PORT dump
```

Connecting...........
Detecting chip type... ESP32
BLOCK0
- 00000000 8e188e44 00ad840d
- 00000000 00000235 00000000 00000005
BLOCK1 (flash_encryption) [1 ] readRegs: 00000000 00000000 00000000
- 00000000 00000000 00000000
BLOCK2 (secure_boot_v1 s) [2 ] readRegs: 00000000 00000000 00000000
- 00000000 00000000 00000000
BLOCK3 [3 ] readRegs: 00000000 00000000 00000000
- fa870291 00000000 00000000
```
esefuse.py v3.1-dev
```

Header File

- components/efuse/esp32/include/esp_efuse_chip.h

Enumerations
enum esp_efuse_block_t
    Type of eFuse blocks for ESP32.

    Values:

    enumerator EFUSE_BLK0
        Number of eFuse block. Reserved.

    enumerator EFUSE_BLK1
        Number of eFuse block. Used for Flash Encryption. If not using that Flash Encryption feature, they can be used for another purpose.

    enumerator EFUSE_BLK_KEY0
        Number of eFuse block. Used for Flash Encryption. If not using that Flash Encryption feature, they can be used for another purpose.

    enumerator EFUSE_BLK_ENCRYPT_FLASH
        Number of eFuse block. Used for Flash Encryption. If not using that Flash Encryption feature, they can be used for another purpose.

    enumerator EFUSE_BLK2
        Number of eFuse block. Used for Secure Boot. If not using that Secure Boot feature, they can be used for another purpose.

    enumerator EFUSE_BLK_KEY1
        Number of eFuse block. Used for Secure Boot. If not using that Secure Boot feature, they can be used for another purpose.

    enumerator EFUSE_BLK_SECURE_BOOT
        Number of eFuse block. Used for Secure Boot. If not using that Secure Boot feature, they can be used for another purpose.

    enumerator EFUSE_BLK3
        Number of eFuse block. Uses for the purpose of the user.

    enumerator EFUSE_BLK_KEY2
        Number of eFuse block. Uses for the purpose of the user.

    enumerator EFUSE_BLK_KEY_MAX

    enumerator EFUSE_BLK_MAX

enum esp_efuse_coding_scheme_t
    Type of coding scheme.

    Values:

    enumerator EFUSE_CODING_SCHEME_NONE
        None
enumerator **EFUSE CODING_SCHEME_3_4**

3/4 coding

eumerator **EFUSE CODING_SCHEME_REPEAT**

Repeat coding

euno **esp_efuse_purpose_t**

Type of key purpose (virtual because ESP32 has only fixed purposes for blocks)

*Values:*

eumerator **ESP_EFUSE_KEY_PURPOSE_USER**

BLOCK3

eumerator **ESP_EFUSE_KEY_PURPOSE_SYSTEM**

BLOCK0

eumerator **ESP_EFUSE_KEY_PURPOSE_FLASH_ENCRYPTION**

BLOCK1

eumerator **ESP_EFUSE_KEY_PURPOSE_SECURE_BOOT_V2**

BLOCK2

eumerator **ESP_EFUSE_KEY_PURPOSE_MAX**

MAX PURPOSE

**Header File**

- components/efuse/include/esp_efuse.h

**Functions**

```c
esp_err_t esp_efuse_read_field_blob(const esp_efuse_desc_t *field[], void *dst, size_t dst_size_bits)
```

Reads bits from EFUSE field and writes it into an array.

The number of read bits will be limited to the minimum value from the description of the bits in “field” structure or “dst_size_bits” required size. Use “esp_efuse_get_field_size()” function to determine the length of the field.

**备注:** Please note that reading in the batch mode does not show uncommitted changes.

**参数**

- **field** – 【in】 A pointer to the structure describing the fields of efuse.
- **dst** – 【out】 A pointer to array that will contain the result of reading.
- **dst_size_bits** – 【in】 The number of bits required to read. If the requested number of bits is greater than the field, the number will be limited to the field size.

**返回**

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
bool esp_efuse_read_field_bit (const esp_efuse_desc_t *field[])  
Read a single bit eFuse field as a boolean value.

备注: The value must exist and must be a single bit wide. If there is any possibility of an error in the provided arguments, call esp Efuse_read_field_blob() and check the returned value instead.

备注: If assertions are enabled and the parameter is invalid, execution will abort.

备注: Please note that reading in the batch mode does not show uncommitted changes.

参数 field –[in] A pointer to the structure describing the fields of efuse.

返回
- true: The field parameter is valid and the bit is set.
- false: The bit is not set, or the parameter is invalid and assertions are disabled.

esp_err_t esp_efuse_read_field_cnt (const esp_efuse_desc_t *field[], size_t*out_cnt)

Reads bits from EFUSE field and returns number of bits programmed as “1”.
If the bits are set not sequentially, they will still be counted.

备注: Please note that reading in the batch mode does not show uncommitted changes.

参数
- field –[in] A pointer to the structure describing the fields of efuse.
- out_cnt –[out] A pointer that will contain the number of programmed as “1” bits.

返回
- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.

esp_err_t esp_efuse_write_field_blob (const esp_efuse_desc_t *field[], const void *src, size_t src_size_bits)

Writes array to EFUSE field.
The number of write bits will be limited to the minimum value from the description of the bits in “field” structure or “src_size_bits” required size. Use “esp_efuse_get_field_size()” function to determine the length of the field. After the function is completed, the writing registers are cleared.

参数
- field –[in] A pointer to the structure describing the fields of efuse.
- src –[in] A pointer to array that contains the data for writing.
- src_size_bits –[in] The number of bits required to write.

返回
- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

esp_err_t esp_efuse_write_field_cnt (const esp_efuse_desc_t *field[], size_t cnt)

Writes a required count of bits as “1” to EFUSE field.
If there are no free bits in the field to set the required number of bits to “1”, ESP_ERR_EFUSE_CNT_IS_FULL error is returned, the field will not be partially recorded. After the function is completed, the writing registers are cleared.
**Chapter 2. API Reference**

### `esp_err_t esp_efuse_write_field_bit (const esp_efuse_desc_t *field[])`

Write a single bit eFuse field to 1.

For use with eFuse fields that are a single bit. This function will write the bit to value 1 if it is not already set, or does nothing if the bit is already set.

This is equivalent to calling `esp_efuse_write_field_cnt()` with the cnt parameter equal to 1, except that it will return ESP_OK if the field is already set to 1.

**Parameters**
- `field` - [in] Pointer to the structure describing the eFuse field.

**Return Values**
- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.

### `esp_err_t esp_efuse_set_write_protect (esp_efuse_block_t blk)`

Sets a write protection for the whole block.

After that, it is impossible to write to this block. The write protection does not apply to block 0.

**Parameters**
- `blk` - [in] Block number of eFuse. (EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3)

**Return Values**
- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.
- ESP_ERR_NOT_SUPPORTED: The block does not support this command.

### `esp_err_t esp_efuse_set_read_protect (esp_efuse_block_t blk)`

Sets a read protection for the whole block.

After that, it is impossible to read from this block. The read protection does not apply to block 0.

**Parameters**
- `blk` - [in] Block number of eFuse. (EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3)

**Return Values**
- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.
- ESP_ERR_NOT_SUPPORTED: The block does not support this command.

### `int esp_efuse_get_field_size (const esp_efuse_desc_t *field[])`

Returns the number of bits used by field.

**Parameters**
- `field` - [in] A pointer to the structure describing the fields of eFuse.

**Return Values**
- Returns the number of bits used by field.

### `uint32_t esp_efuse_read_reg (esp_efuse_block_t blk, unsigned int num_reg)`

Returns value of eFuse register.

This is a thread-safe implementation. Example: EFUSE_BLK2_RDATA3_REG where (blk=2, num_reg=3)

**Parameters**
- `blk` - [in] Block number of eFuse.

---

**Notes:** Please note that reading in the batch mode does not show uncommitted changes.
Chapter 2. API リファレンス

- **num_reg** - [in] The register number in the block.

**esp_err_t esp_efuse_write_reg** (esp_efuse_block_t blk, unsigned int num_reg, uint32_t val)

Write value to efuse register.

Apply a coding scheme if necessary. This is a thread-safe implementation. Example: EFUSE_BLK3_WDATA0_REG where (blk=3, num_reg=0)

**Parameters**
- **blk** - [in] Block number of EFuse.
- **num_reg** - [in] The register number in the block.
- **val** - [in] Value to write.

**Return**
- **ESP_OK**: The operation was successfully completed.
- **ESP_ERR_EFUSE_REPEATED_PROG**: Error repeated programming of programmed bits is strictly forbidden.

**esp_efuse_coding_scheme_t esp_efuse_get_coding_scheme** (esp_efuse_block_t blk)

Return efuse coding scheme for blocks.

Note: The coding scheme is applicable only to 1, 2 and 3 blocks. For 0 block, the coding scheme is always NONE.

**Parameters**
- **blk** - [in] Block number of EFuse.

**Return**
- Return efuse coding scheme for blocks

**esp_err_t esp_efuse_read_block** (esp_efuse_block_t blk, void *dst_key, size_t offset_in_bits, size_t size_bits)

Read key to efuse block starting at the offset and the required size.

**Note:** Please note that reading in the batch mode does not show uncommitted changes.

**Parameters**
- **blk** - [in] Block number of EFuse.
- **dst_key** - [in] A pointer to array that will contain the result of reading.
- **size_bits** - [in] The number of bits required to read.

**Return**
- **ESP_OK**: The operation was successfully completed.
- **ESP_ERR_INVALID_ARG**: Error in the passed arguments.
- **ESP_ERR_CODING**: Error range of data does not match the coding scheme.

**esp_err_t esp_efuse_write_block** (esp_efuse_block_t blk, const void *src_key, size_t offset_in_bits, size_t size_bits)

Write key to efuse block starting at the offset and the required size.

**Parameters**
- **blk** - [in] Block number of EFuse.
- **src_key** - [in] A pointer to array that contains the key for writing.
- **size_bits** - [in] The number of bits required to write.

**Return**
- **ESP_OK**: The operation was successfully completed.
- **ESP_ERR_INVALID_ARG**: Error in the passed arguments.
- **ESP_ERR_CODING**: Error range of data does not match the coding scheme.
- **ESP_ERR_EFUSE_REPEATED_PROG**: Error repeated programming of programmed bits.
uint32_t esp_efuse_get_pkg_ver (void)

Returns chip package from efuse.

void esp_efuse_reset (void)

Reset efuse write registers.

Efuse write registers are written to zero, to negate any changes that have been staged here.

备注: This function is not threadsafe, if calling code updates efuse values from multiple tasks then this is caller’s responsibility to serialize.

void esp_efuse_disable_basic_rom_console (void)

Disable BASIC ROM Console via efuse.

By default, if booting from flash fails the ESP32 will boot a BASIC console in ROM.

Call this function (from bootloader or app) to permanently disable the console on this chip.

esp_err_t esp_efuse_disable_rom_download_mode (void)

Disable ROM Download Mode via eFuse.

Permanently disables the ROM Download Mode feature. Once disabled, if the SoC is booted with strapping pins set for ROM Download Mode then an error is printed instead.

备注: Not all SoCs support this option. An error will be returned if called on an ESP32 with a silicon revision lower than 3, as these revisions do not support this option.

备注: If ROM Download Mode is already disabled, this function does nothing and returns success.

返回

• ESP_OK If the eFuse was successfully burned, or had already been burned.
• ESP_ERR_NOT_SUPPORTED (ESP32 only) This SoC is not capable of disabling UART download mode
• ESP_ERR_INVALID_STATE (ESP32 only) This eFuse is write protected and cannot be written

esp_err_t esp_efuse_set_rom_log_scheme (esp_efuse_rom_log_scheme_t log_scheme)

Set boot ROM log scheme via eFuse.

备注: By default, the boot ROM will always print to console. This API can be called to set the log scheme only once per chip, once the value is changed from the default it can’t be changed again.

参数 log_scheme – Supported ROM log scheme

返回

• ESP_OK If the eFuse was successfully burned, or had already been burned.
• ESP_ERR_NOT_SUPPORTED (ESP32 only) This SoC is not capable of setting ROM log scheme
• ESP_ERR_INVALID_STATE This eFuse is write protected or has been burned already

uint32_t esp_efuse_read_secure_version (void)

Return secure_version from efuse field.

返回 Secure version from efuse field
bool esp_efuse_check_secure_version (uint32_t secure_version)

Check secure_version from app and secure_version and from efuse field.

  参数 secure_version – Secure version from app.
  返回
  • True: If version of app is equal or more then secure_version from efuse.

esp_err_t esp_efuse_update_secure_version (uint32_t secure_version)

Write efuse field by secure_version value.

Update the secure_version value is available if the coding scheme is None. Note: Do not use this function in your applications. This function is called as part of the other API.

  参数 secure_version –[in] Secure version from app.
  返回
  • ESP_OK: Successful.
  • ESP_FAIL: secure version of app cannot be set to efuse field.
  • ESP_ERR_NOT_SUPPORTED: Anti rollback is not supported with the 3/4 and Repeat coding scheme.

esp_err_t esp_efuse_batch_write_begin (void)

Set the batch mode of writing fields.

This mode allows you to write the fields in the batch mode when need to burn several efuses at one time. To enable batch mode call begin() then perform as usually the necessary operations read and write and at the end call commit() to actually burn all written efuses. The batch mode can be used nested. The commit will be done by the last commit() function. The number of begin() functions should be equal to the number of commit() functions.

Note: If batch mode is enabled by the first task, at this time the second task cannot write/read efuses. The second task will wait for the first task to complete the batch operation.

```
// Example of using the batch writing mode.

// set the batch writing mode
esp_efuse_batch_write_begin();

// use any writing functions as usual
esp_efuse_write_field_blob(ESP_EFUSE_...);
esp_efuse_write_field_cnt(ESP_EFUSE_...);
esp_efuse_set_write_protect(EFUSE_BLKx);
esp_efuse_write_reg(EFUSE_BLKx, ...);
esp_efuse_write_block(EFUSE_BLKx, ...);
esp_efuse_write(ESP_EFUSE_1, 3); // ESP_EFUSE_1 == 1, here we write a new...
                               // value = 3. The changes will be burn by the commit() function.
esp_efuse_read...(ESP_EFUSE_1); // this function returns ESP_EFUSE_1 == 1...
                              // because uncommitted changes are not readable, it will be available only...
                              // after commit.
...
// esp_efuse_batch_write APIs can be called recursively.
esp_efuse_batch_write_begin();
esp_efuse_set_write_protect(EFUSE_BLKx);
esp_efuse_batch_write_commit(); // the burn will be skipped here, it will be...
                                 // done in the last commit().
...
// Write all of these fields to the efuse registers
```
Chapter 2. API

```c
esp_efuse_batch_write_commit();
esp_efuse_read_...(ESP_EFUSE_1);  // this function returns ESP_EFUSE_1 == 3.
```

备注：Please note that reading in the batch mode does not show uncommitted changes.

返回
- ESP_OK: Successful.

**esp_err_t esp_efuse_batch_write_cancel**(void)

Reset the batch mode of writing fields.

It will reset the batch writing mode and any written changes.

返回
- ESP_OK: Successful.
- ESP_ERR_INVALID_STATE: The batch mode was not set.

**esp_err_t esp_efuse_batch_write_commit**(void)

Writes all prepared data for the batch mode.

Must be called to ensure changes are written to the efuse registers. After this the batch writing mode will be reset.

返回
- ESP_OK: Successful.
- ESP_ERR_INVALID_STATE: The deferred writing mode was not set.

**bool esp_efuse_block_is_empty**(esp_efuse_block_t block)

Checks that the given block is empty.

返回
- True: The block is empty.
- False: The block is not empty or was an error.

**bool esp_efuse_get_key_dis_read**(esp_efuse_block_t block)

Returns a read protection for the key block.

参数 block -[in] A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX

返回 True: The key block is read protected False: The key block is readable.

**esp_err_t esp_efuse_set_key_dis_read**(esp_efuse_block_t block)

Sets a read protection for the key block.

参数 block -[in] A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX

返回
- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

**bool esp_efuse_get_key_dis_write**(esp_efuse_block_t block)

Returns a write protection for the key block.

参数 block -[in] A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX

返回 True: The key block is write protected False: The key block is writeable.

**esp_err_t esp_efuse_set_key_dis_write**(esp_efuse_block_t block)

Sets a write protection for the key block.

参数 block -[in] A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
Chapter 2. API 参考

返回
- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed
  bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

bool esp_efuse_key_block_unused(esp_efuse_block_t block)

Returns true if the key block is unused, false otherwise.

参数 block – key block to check.

返回
- True if key block is unused,
- False if key block is used or the specified block index is not a key block.

bool esp_efuse_find_purpose(esp_efuse_purpose_t purpose, esp_efuse_block_t *block)

Find a key block with the particular purpose set.

参数
- purpose – [in] Purpose to search for.
- block – [out] Pointer in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX
  which will be set to the key block if found. Can be NULL, if only need to test the key
  block exists.

返回
- True: If found,
- False: If not found (value at block pointer is unchanged).

bool esp_efuse_get_keypurpose_dis_write(esp_efuse_block_t block)

Returns a write protection of the key purpose field for an efuse key block.

备注: For ESP32: no keypurpose, it returns always True.

参数 block – [in] A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX

返回 True: The key purpose is write protected. False: The key purpose is writeable.

esp_efuse_purpose_t esp_efuse_get_key_purpose(esp_efuse_block_t block)

Returns the current purpose set for an efuse key block.

参数 block – [in] A key block in the range EFUSE_BLK_KEY0..EFUSE_BLK_KEY_MAX

返回
- Value: If Successful, it returns the value of the purpose related to the given key block.
- ESP_EFUSE_KEY_PURPOSE_MAX: Otherwise.

esp_err_t esp_efuse_write_key(esp_efuse_block_t block, esp_efuse_purpose_t purpose, const void *key,
  size_t key_size_bytes)

Program a block of key data to an efuse block.

The burn of a key, protection bits, and a purpose happens in batch mode.

参数
- block – [in] Block to read purpose for. Must be in range EFUSE_BLK_KEY0 to
  EFUSE_BLK_KEY_MAX. Key block must be unused (esp_efuse_key_block_unused).
- purpose – [in] Purpose to set for this key. Purpose must be already unset.
- key – [in] Pointer to data to write.
- key_size_bytes – [in] Bytes length of data to write.

返回
- ESP_OK: Successful.
Chapter 2. API

- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_INVALID_STATE: Error in efuses state, unused block not found.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

`esp_err_t esp_efuse_write_keys (const esp_efuse_purpose_t purposes[], uint8_t keys[][32], unsigned number_of_keys)`

Program keys to unused efuse blocks.

The burn of keys, protection bits, and purposes happens in batch mode.

参数
- `purposes` - [in] Array of purposes (purpose[number_of_keys]).
- `keys` - [in] Array of keys (uint8_t keys[number_of_keys][32]). Each key is 32 bytes long.
- `number_of_keys` - [in] The number of keys to write (up to 6 keys).

返回
- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_INVALID_STATE: Error in efuses state, unused block not found.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

`esp_err_t esp_efuse_check_errors (void)`

Checks eFuse errors in BLOCK0.

It does a BLOCK0 check if eFuse EFUSE_ERR_RST_ENABLE is set. If BLOCK0 has an error, it prints the error and returns ESP_FAIL, which should be treated as esp_restart.

备注：Refer to ESP32-C3 only.

返回
- ESP_OK: No errors in BLOCK0.
- ESP_FAIL: Error in BLOCK0 requiring reboot.

Structures

`struct esp_efuse_desc_t`  
Type definition for an eFuse field.

Public Members

`esp_efuse_block_t efuse_block`  
Block of eFuse

`uint8_t bit_start`  
Start bit [0..255]

`uint16_t bit_count`  
Length of bit field [1..-]
Chapter 2. API

Macros

ESP_ERR_EFUSE
Base error code for efuse api.

ESP_OK_EFUSE_CNT
OK the required number of bits is set.

ESP_ERR_EFUSE_CNT_IS_FULL
Error field is full.

ESP_ERR_EFUSE_REPEATED_PROG
Error repeated programming of programmed bits is strictly forbidden.

ESP_ERR_CODING
Error while a encoding operation.

ESP_ERR_NOT_ENOUGH_UNUSED_KEY_BLOCKS
Error not enough unused key blocks available

ESP_ERR_DAMAGED_READING
Error. Burn or reset was done during a reading operation leads to damage read data. This error is internal to
the efuse component and not returned by any public API.

Enumerations

enum esp_efuse_rom_log_scheme_t
Type definition for ROM log scheme.

Values:

enumerator ESP_EFUSE_ROM_LOG_ALWAYS_ON
Always enable ROM logging

enumerator ESP_EFUSE_ROM_LOG_ON_GPIO_LOW
ROM logging is enabled when specific GPIO level is low during start up

enumerator ESP_EFUSE_ROM_LOG_ON_GPIO_HIGH
ROM logging is enabled when specific GPIO level is high during start up

enumerator ESP_EFUSE_ROM_LOG_ALWAYS_OFF
Disable ROM logging permanently

2.10.7 Error Codes and Helper Functions

This section lists definitions of common ESP-IDF error codes and several helper functions related to error handling.
For general information about error codes in ESP-IDF, see Error Handling.
For the full list of error codes defined in ESP-IDF, see Error Code Reference.
Chapter 2. API 参考

API Reference

Header File

- components/esp_common/include/esp_check.h

Macros

ESP_RETURN_ON_ERROR(x, log_tag, format, ...)

Macro which can be used to check the error code. If the code is not ESP_OK, it prints the message and returns. In the future, we want to switch to C++20. We also want to become compatible with clang. Hence, we provide two versions of the following macros. The first one is using the GNU extension ##__VA_ARGS__. The second one is using the C++20 feature VA_OPT(.). This allows users to compile their code with standard C++20 enabled instead of the GNU extension. Below C++20, we haven’t found any good alternative to using ##__VA_ARGS__. Macro which can be used to check the error code. If the code is not ESP_OK, it prints the message and returns.

ESP_RETURN_ON_ERROR_ISR(x, log_tag, format, ...)

A version of ESP_RETURN_ON_ERROR() macro that can be called from ISR.

ESP_GOTO_ON_ERROR(x, goto_tag, log_tag, format, ...)

Macro which can be used to check the error code. If the code is not ESP_OK, it prints the message, sets the local variable ‘ret’ to the code, and then exits by jumping to ‘goto_tag’.

ESP_GOTO_ON_ERROR_ISR(x, goto_tag, log_tag, format, ...)

A version of ESP_GOTO_ON_ERROR() macro that can be called from ISR.

ESP_RETURN_ON_FALSE(a, err_code, log_tag, format, ...)

Macro which can be used to check the condition. If the condition is not ‘true’, it prints the message and returns with the supplied ‘err_code’.

ESP_RETURN_ON_FALSE_ISR(a, err_code, goto_tag, log_tag, format, ...)

A version of ESP_RETURN_ON_FALSE() macro that can be called from ISR.

ESP_GOTO_ON_FALSE(a, err_code, goto_tag, log_tag, format, ...)

Macro which can be used to check the condition. If the condition is not ‘true’, it prints the message, sets the local variable ‘ret’ to the supplied ‘err_code’, and then exits by jumping to ‘goto_tag’.

ESP_GOTO_ON_FALSE_ISR(a, err_code, goto_tag, log_tag, format, ...)

A version of ESP_GOTO_ON_FALSE() macro that can be called from ISR.

Header File

- components/esp_common/include/esp_err.h

Functions

const char *esp_err_to_name(esp_err_t code)

Returns string for esp_err_t error codes.

This function finds the error code in a pre-generated lookup-table and returns its string representation.

The function is generated by the Python script tools/gen_esp_err_to_name.py which should be run each time an esp_err_t error is modified, created or removed from the IDF project.

参数 code - esp_err_t error code
返回 string error message

const char *esp_err_to_name_r(esp_err_t code, char *buf, size_t buflen)

Returns string for esp_err_t and system error codes.

This function finds the error code in a pre-generated lookup-table of esp_err_t errors and returns its string representation. If the error code is not found then it is attempted to be found among system errors.
The function is generated by the Python script tools/gen_esp_err_to_name.py which should be run each time an esp_err_t error is modified, created or removed from the IDF project.

参数
- **code** - esp_err_t error code
- **buf** - [out] buffer where the error message should be written
- **buflen** - Size of buffer buf. At most buflen bytes are written into the buf buffer (including the terminating null byte).

返回 buf containing the string error message

**Macros**

ESP_OK  
esp_err_t value indicating success (no error)

ESP_FAIL  
Generic esp_err_t code indicating failure

ESP_ERR_NO_MEM  
Out of memory

ESP_ERR_INVALID_ARG  
Invalid argument

ESP_ERR_INVALID_STATE  
Invalid state

ESP_ERR_INVALID_SIZE  
Invalid size

ESP_ERR_NOT_FOUND  
Requested resource not found

ESP_ERR_NOT_SUPPORTED  
Operation or feature not supported

ESP_ERR_TIMEOUT  
Operation timed out

ESP_ERR_INVALID_RESPONSE  
Received response was invalid

ESP_ERR_INVALID_CRC  
CRC or checksum was invalid

ESP_ERR_INVALID_VERSION  
Version was invalid

ESP_ERR_INVALID_MAC  
MAC address was invalid
**Chapter 2. API 参考**

**ESP_ERR_NOT_FINISHED**
There are items remained to retrieve

**ESP_ERR_WIFI_BASE**
Starting number of WiFi error codes

**ESP_ERR_MESH_BASE**
Starting number of MESH error codes

**ESP_ERR_FLASH_BASE**
Starting number of flash error codes

**ESP_ERR_HW_CRYPTO_BASE**
Starting number of HW cryptography module error codes

**ESP_ERR_MEMPROT_BASE**
Starting number of Memory Protection API error codes

**ESP_ERROR_CHECK (x)**
Macro which can be used to check the error code, and terminate the program in case the code is not ESP_OK.
Prints the error code, error location, and the failed statement to serial output.
Disabled if assertions are disabled.

**ESP_ERROR_CHECK_WITHOUT_ABORT (x)**
Macro which can be used to check the error code. Prints the error code, error location, and the failed statement to serial output. In comparison with ESP_ERROR_CHECK(), this prints the same error message but isn’t terminating the program.

**Type Definitions**
typedef int esp_err_t

### 2.10.8 ESP HTTPS OTA

**Overview**

`esp_https_ota` provides simplified APIs to perform firmware upgrades over HTTPS. It’s an abstraction layer over existing OTA APIs.

**Application Example**

```c
esp_err_t do_firmware_upgrade() {
    esp_http_client_config_t config = {
        .url = CONFIG_FIRMWARE_UPGRADE_URL,
        .cert_pem = (char *)server_cert_pem_start,
    };
    esp_https_ota_config_t ota_config = {
        .http_config = &config,
    };
    esp_err_t ret = esp_https_ota(&ota_config);
    if (ret == ESP_OK) {
```
Server Verification

Please refer to ESP-TLS: TLS Server Verification for more information on server verification. The root certificate (in PEM format) needs to be provided to the esp_http_client_config_t::cert_pem member.

备注：The server-endpoint root certificate should be used for verification instead of any intermediate ones from the certificate chain. The reason being that the root certificate has the maximum validity and usually remains the same for a long period of time. Users can also use the ESP x509 Certificate Bundle feature for verification, which covers most of the trusted root certificates (using the esp_http_client_config_t::crt_bundle_attach member).

Partial Image Download over HTTPS

To use partial image download feature, enable partial_http_download configuration in esp_https_ota_config_t. When this configuration is enabled, firmware image will be downloaded in multiple HTTP requests of specified size. Maximum content length of each request can be specified by setting max_http_request_size to required value.

This option is useful while fetching image from a service like AWS S3, where mbedTLS Rx buffer size (CONFIG_MBEDTLS_SSL_IN_CONTENT_LEN) can be set to lower value which is not possible without enabling this configuration.

Default value of mbedTLS Rx buffer size is set to 16K. By using partial_http_download with max_http_request_size of 4K, size of mbedTLS Rx buffer can be reduced to 4K. With this configuration, memory saving of around 12K is expected.

Signature Verification

For additional security, signature of OTA firmware images can be verified. For that, refer 没有安全启动的安全OTA升级.

Advanced APIs

esp=https_ota also provides advanced APIs which can be used if more information and control is needed during the OTA process.

Example that uses advanced ESP_HTTPS OTA APIs: system/ota/advanced_https_ota.

OTA Upgrades with Pre-Encrypted Firmware

To perform OTA upgrades with Pre-Encrypted Firmware, please enable CONFIG_ESP_HTTPS_OTA_DECRYPT_CB in component menuconfig.

Example that performs OTA upgrade with Pre-Encrypted Firmware: system/ota/pre_encrypted_ota.
Chapter 2. API 参考

API Reference

Header File

• components/esp_https_ota/include/esp_https_ota.h

Functions

esp_err_t esp_https_ota (const esp_https_ota_config_t *ota_config)

HTTPS OTA Firmware upgrade.

This function allocates HTTPS OTA Firmware upgrade context, establishes HTTPS connection, reads image data from HTTP stream and writes it to OTA partition and finishes HTTPS OTA Firmware upgrade operation. This API supports URL redirection, but if CA cert of URLs differ then it should be appended to cert.pem member of ota_config->http_config.

备注： This API handles the entire OTA operation, so if this API is being used then no other APIs from esp_https_ota component should be called. If more information and control is needed during the HTTPS OTA process, then one can use esp_https_ota_begin and subsequent APIs. If this API returns successfully, esp_restart() must be called to boot from the new firmware image.

参数 ota_config [in] pointer to esp_https_ota_config_t structure.

返回

• ESP_OK: OTA data updated, next reboot will use specified partition.
• ESP_FAIL: For generic failure.
• ESP_ERR_INVALID_ARG: Invalid argument
• ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
• ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
• ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
• For other return codes, refer OTA documentation in esp-idf’s app_update component.

esp_err_t esp_https_ota_begin (const esp_https_ota_config_t *ota_config, esp_https_ota_handle_t *handle)

Start HTTPS OTA Firmware upgrade.

This function initializes ESP HTTPS OTA context and establishes HTTPS connection. This function must be invoked first. If this function returns successfully, then esp_https_ota_perform should be called to continue with the OTA process and there should be a call to esp_https_ota_finish on completion of OTA operation or on failure in subsequent operations. This API supports URL redirection, but if CA cert of URLs differ then it should be appended to cert.pem member of http_config, which is a part of ota_config. In case of error, this API explicitly sets handle to NULL.

备注： This API is blocking, so setting is_async member of http_config structure will result in an error.

参数 ota_config [in] pointer to esp_https_ota_config_t structure

handle [out] pointer to an allocated data of type esp_https_ota_handle_t which will be initialised in this function

返回

• ESP_OK: HTTPS OTA Firmware upgrade context initialised and HTTPS connection established
• ESP_FAIL: For generic failure.
• ESP_ERR_INVALID_ARG: Invalid argument (missing/incorrect config, certificate, etc.)
• For other return codes, refer documentation in app_update component and esp_http_client component in esp-idf.
**esp_err_t esp_https_ota_perform** *(esp_https_ota_handle_t https_ota_handle)*

Read image data from HTTP stream and write it to OTA partition.

This function reads image data from HTTP stream and writes it to OTA partition. This function must be called only if esp_https_ota_begin() returns successfully. This function must be called in a loop since it returns after every HTTP read operation thus giving you the flexibility to stop OTA operation midway.

参数 https_ota_handle — [in] pointer to esp_https_ota_handle_t structure

返回

- ESP_ERR_HTTPS_OTA_IN_PROGRESS: OTA update is in progress, call this API again to continue.
- ESP_OK: OTA update was successful
- ESP_FAIL: OTA update failed
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_INVALID_VERSION: Invalid chip revision in image header
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf’s app_update component.

**bool esp_https_ota_is_complete_data_received** *(esp_https_ota_handle_t https_ota_handle)*

Checks if complete data was received or not.

备注: This API can be called just before esp_https_ota_finish() to validate if the complete image was indeed received.

参数 https_ota_handle — [in] pointer to esp_https_ota_handle_t structure

返回

- false
- true

**esp_err_t esp_https_ota_finish** *(esp_https_ota_handle_t https_ota_handle)*

Clean-up HTTPS OTA Firmware upgrade and close HTTPS connection.

This function closes the HTTP connection and frees the ESP HTTPS OTA context. This function switches the boot partition to the OTA partition containing the new firmware image.

备注: If this API returns successfully, esp_restart() must be called to boot from the new firmware image esp_https_ota_finish should not be called after calling esp_https_ota_abort

参数 https_ota_handle — [in] pointer to esp_https_ota_handle_t structure

返回

- ESP_OK: Clean-up successful
- ESP_ERR_INVALID_STATE
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image

**esp_err_t esp_https_ota_abort** *(esp_https_ota_handle_t https_ota_handle)*

Clean-up HTTPS OTA Firmware upgrade and close HTTPS connection.

This function closes the HTTP connection and frees the ESP HTTPS OTA context.

备注: esp_https_ota_abort should not be called after calling esp_https_ota_finish
Chapter 2. API

### Parameters

**https_ota_handle** - [in] pointer to esp_https_ota_handle_t structure

**new_app_info** - [out] pointer to an allocated esp_app_desc_t structure

### Return

- ESP_OK: Clean-up successful
- ESP_ERR_INVALID_STATE: Invalid ESP HTTPS OTA state
- ESP_FAIL: OTA not started
- ESP_ERR_NOT_FOUND: OTA handle not found
- ESP_ERR_INVALID_ARG: Invalid argument

### Remarks

This API can be called only after `esp_https_ota_begin()` and before `esp_https_ota_perform()`. Calling this API is not mandatory.

### esp_err_t esp_https_ota_get_img_desc

```c

| Structures |
|---|---|
| **esp_https_ota_config_t** | ESP HTTPS OTA configuration. |
Chapter 2. API 参考

Public Members

const esp_http_client_config_t *http_config
ESP HTTP client configuration

http_client_init_cb_t http_client_init_cb
Callback after ESP HTTP client is initialised

bool bulk_flash_erase
Erase entire flash partition during initialization. By default flash partition is erased during write operation and in chunk of 4K sector size

bool partial_http_download
Enable Firmware image to be downloaded over multiple HTTP requests

int max_http_request_size
Maximum request size for partial HTTP download

Macros

ESP_ERR_HTTPS_OTA_BASE
ESP_ERR_HTTPS_OTA_IN_PROGRESS

Type Definitions

typedef void *esp_https_ota_handle_t

typedef esp_err_t (*http_client_init_cb_t)(esp_http_client_handle_t)

Enumerations

enum esp_https_ota_event_t
Events generated by OTA process.

备注: Expected data type for different OTA events:

- ESP_HTTPS_OTA_START : NULL
- ESP_HTTPS_OTA_CONNECTED : NULL
- ESP_HTTPS_OTA_GET_IMG_DESC : NULL
- ESP_HTTPS_OTA_VERIFY_CHIP_ID : esp_chip_id_t
- ESP_HTTPS_OTA_DECRYPT_CB : NULL
- ESP.Https_OTA_WRITE_FLASH : int
- ESP_HTTPS_OTA_UPDATE_BOOT_PARTITION : esp_partition_subtype_t
- ESP_HTTPS_OTA_FINISH : NULL
- ESP_HTTPS_OTA_ABORT : NULL

Values:

enumerator ESP_HTTPS_OTA_START
OTA started
**Chapter 2. API**

**Enumerator**: ESP_HTTPS_OTA_CONNECTED

*Connected to server*

**Enumerator**: ESP_HTTPS_OTA_GET_IMG_DESC

*Read app description from image header*

**Enumerator**: ESP_HTTPS_OTA_VERIFY_CHIP_ID

*Verify chip id of new image*

**Enumerator**: ESP_HTTPS_OTA_DECRYPT_CB

*Callback to decrypt function*

**Enumerator**: ESP_HTTPS_OTA_WRITE_FLASH

*Flash write operation*

**Enumerator**: ESP_HTTPS_OTA_UPDATE_BOOT_PARTITION

*Boot partition update after successful ota update*

**Enumerator**: ESP_HTTPS_OTA_FINISH

*OTA finished*

**Enumerator**: ESP_HTTPS_OTA_ABORT

*OTA aborted*

## 2.10.9 Event Loop Library

**Overview**

The event loop library allows components to declare events to which other components can register handlers that will execute when those events occur. This allows loosely coupled components to attach desired behavior to changes in state of other components without application involvement. For instance, a high level connection handling library may subscribe to events produced by the Wi-Fi subsystem directly and act on those events. This also simplifies event processing by serializing and deferring code execution to another context.

### Using esp_event APIs

There are two objects of concern for users of this library: events and event loops.

- Events are occurrences of note. For example, for Wi-Fi, a successful connection to the access point may be an event. Events are referenced using a two part identifier which are discussed more here. Event loops are the vehicle by which events get posted by event sources and handled by event handler functions. These two appear prominently in the event loop library APIs.

Using this library roughly entails the following flow:

1. A user defines a function that should run when an event is posted to a loop. This function is referred to as the event handler. It should have the same signature as esp_event_handler_t.
2. An event loop is created using esp_event_loop_create(), which outputs a handle to the loop of type esp_event_loop_handle_t. Event loops created using this API are referred to as user event loops. There is, however, a special type of event loop called the default event loop which are discussed here.
3. Components register event handlers to the loop using esp_event_handler_register_with(). Handlers can be registered with multiple loops, more on that here.
4. Event sources post an event to the loop using esp_event_post_to().
5. Components wanting to remove their handlers from being called can do so by unregistering from the loop using `esp_event_handler_unregister_with()`.

6. Event loops which are no longer needed can be deleted using `esp_event_loop_delete()`.

In code, the flow above may look like as follows:

```c
// 1. Define the event handler
void run_on_event(void* handler_arg, esp_event_base_t base, int32_t id, void* event_data) {
    // Event handler logic
}

void app_main() {
    // 2. A configuration structure of type esp_event_loop_args_t is needed to specify the properties of the loop to be created. A handle of type esp_event_loop_handle_t is obtained, which is needed by the other APIs to reference the loop to perform their operations on.
    esp_event_loop_args_t loop_args = {
        .queue_size = ...,
        .task_name = ...,
        .task_priority = ...,
        .task_stack_size = ...,
        .task_core_id = ...
    };

    esp_event_loop_handle_t loop_handle;
    esp_event_loop_create(&loop_args, &loop_handle);

    // 3. Register event handler defined in (1). MY_EVENT_BASE and MY_EVENT_ID specifies a hypothetical event that handler run_on_event should execute on when it gets posted to the loop.
    esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_on_event, ...);
    ...

    // 4. Post events to the loop. This queues the event on the event loop. At some point in time the event loop executes the event handler registered to the posted event, in this case run_on_event.
    // For simplicity sake this example calls esp_event_post_to from app_main, but posting can be done from any other tasks (which is the more interesting use case).
    esp_event_post_to(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, ...);
    ...

    // 5. Unregistering an unneeded handler
    esp_event_handler_unregister_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_on_event);
    ...

    // 6. Deleting an unneeded event loop
    esp_event_loop_delete(loop_handle);
}
```
Declaring and defining events

As mentioned previously, events consists of two-part identifiers: the event base and the event ID. The event base identifies an independent group of events; the event ID identifies the event within that group. Think of the event base and event ID as a person’s last name and first name, respectively. A last name identifies a family, and the first name identifies a person within that family.

The event loop library provides macros to declare and define the event base easily.

Event base declaration:

```
ESP_EVENT_DECLARE_BASE(EVENT_BASE)
```

Event base definition:

```
ESP_EVENT_DEFINE_BASE(EVENT_BASE)
```

备注： In IDF, the base identifiers for system events are uppercase and are postfixed with _EVENT. For example, the base for Wi-Fi events is declared and defined as WIFI_EVENT, the ethernet event base ETHERNET_EVENT, and so on. The purpose is to have event bases look like constants (although they are global variables considering the definitions of macros ESP_EVENT_DECLARE_BASE and ESP_EVENTDEFINE_BASE).

For event ID’s, declaring them as enumerations is recommended. Once again, for visibility, these are typically placed in public header files.

Event ID:

```
enum {
    EVENT_ID_1,
    EVENT_ID_2,
    EVENT_ID_3,
    ...
}
```

Default Event Loop

The default event loop is a special type of loop used for system events (Wi-Fi events, for example). The handle for this loop is hidden from the user. The creation, deletion, handler registration/unregistration and posting of events is done through a variant of the APIs for user event loops. The table below enumerates those variants, and the user event loops equivalent.

<table>
<thead>
<tr>
<th>User Event Loops</th>
<th>Default Event Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>esp_event_loop_create()</td>
<td>esp_event_loop_create_default()</td>
</tr>
<tr>
<td>esp_event_loop_delete()</td>
<td>esp_event_loop_delete_default()</td>
</tr>
<tr>
<td>esp_event_handler_register_with()</td>
<td>esp_event_handler_register()</td>
</tr>
<tr>
<td>esp_event_handler_unregister_with()</td>
<td>esp_event_handler_unregister()</td>
</tr>
<tr>
<td>esp_event_post_to()</td>
<td>esp_event_post()</td>
</tr>
</tbody>
</table>

If you compare the signatures for both, they are mostly similar except the for the lack of loop handle specification for the default event loop APIs.

Other than the API difference and the special designation to which system events are posted to, there is no difference to how default event loops and user event loops behave. It is even possible for users to post their own events to the default event loop, should the user opt to not create their own loops to save memory.
Notes on Handler Registration

It is possible to register a single handler to multiple events individually, i.e., using multiple calls to `esp_event_handler_register_with()`. For those multiple calls, the specific event base and event ID can be specified with which the handler should execute.

However, in some cases it is desirable for a handler to execute on (1) all events that get posted to a loop or (2) all events of a particular base identifier. This is possible using the special event base identifier `ESP_EVENT_ANY_BASE` and special event ID `ESP_EVENT_ANY_ID`. These special identifiers may be passed as the event base and event ID arguments for `esp_event_handler_register_with()`.

Therefore, the valid arguments to `esp_event_handler_register_with()` are:

1. `<event base>, <event ID>` - handler executes when the event with base `<event base>` and event ID `<event ID>` gets posted to the loop
2. `<event base>, ESP_EVENT_ANY_ID` - handler executes when any event with base `<event base>` gets posted to the loop
3. `ESP_EVENT_ANY_BASE, ESP_EVENT_ANY_ID` - handler executes when any event gets posted to the loop

As an example, suppose the following handler registrations were performed:

```c
esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_on_event_1, ...);
esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, ESP_EVENT_ANY_ID, run_on_event_2, ...);
esp_event_handler_register_with(loop_handle, ESP_EVENT_ANY_BASE, ESP_EVENT_ANY_ID, run_on_event_3, ...);
```

If the hypothetical event `MY_EVENT_BASE, MY_EVENT_ID` is posted, all three handlers `run_on_event_1`, `run_on_event_2`, and `run_on_event_3` would execute.

If the hypothetical event `MY_EVENT_BASE, MY_OTHER_EVENT_ID` is posted, only `run_on_event_2` and `run_on_event_3` would execute.

If the hypothetical event `MY_OTHER_EVENT_BASE, MY_OTHER_EVENT_ID` is posted, only `run_on_event_3` would execute.

**Handler Registration and Handler Dispatch Order**

The general rule is that for handlers that match a certain posted event during dispatch, those which are registered first also get executed first. The user can then control which handlers get executed first by registering them before other handlers, provided that all registrations are performed using a single task. If the user plans to take advantage of this behavior, caution must be exercised if there are multiple tasks registering handlers. While the ‘first registered, first executed’ behavior still holds true, the task which gets executed first will also get their handlers registered first. Handlers registered one after the other by a single task will still be dispatched in the order relative to each other, but if that task gets pre-empted in between registration by another task which also registers handlers; then during dispatch those handlers will also get executed in between.

**Event loop profiling**

A configuration option `CONFIG_ESP_EVENT_LOOP_PROFILING` can be enabled in order to activate statistics collection for all event loops created. The function `esp_event_dump()` can be used to output the collected statistics to a file stream. More details on the information included in the dump can be found in the `esp_event_dump()` API Reference.

**Application Example**

Examples on using the `esp_event` library can be found in `system/esp_event`. The examples cover event declaration, loop creation, handler registration and unregistration and event posting.

Other examples which also adopt `esp_event` library:
Chapter 2. API Reference

- NMEA Parser, which will decode the statements received from GPS.

**Header File**

- components/esp_event/include/esp_event.h

**Functions**

```c
esp_err_t esp_event_loop_create(const esp_event_loop_args_t *event_loop_args, esp_event_loop_handle_t *event_loop)
```

Create a new event loop.

- `event_loop_args` - [in] configuration structure for the event loop to create
- `event_loop` - [out] handle to the created event loop

- **Return**
  - ESP_OK: Success
  - ESP_ERR_INVALID_ARG: `event_loop_args` or `event_loop` was NULL
  - ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
  - ESP_FAIL: Failed to create task loop
  - Others: Fail

```c
esp_err_t esp_event_loop_delete(esp_event_loop_handle_t event_loop)
```

Delete an existing event loop.

- `event_loop` - [in] event loop to delete, must not be NULL

- **Return**
  - ESP_OK: Success
  - Others: Fail

```c
esp_err_t esp_event_loop_create_default (void)
```

Create default event loop.

- **Return**
  - ESP_OK: Success
  - ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
  - ESP_FAIL: Failed to create task loop
  - Others: Fail

```c
esp_err_t esp_event_loop_delete_default (void)
```

Delete the default event loop.

- **Return**
  - ESP_OK: Success
  - Others: Fail

```c
esp_err_t esp_event_loop_run (esp_event_loop_handle_t event_loop, TickType_t ticks_to_run)
```

Dispatch events posted to an event loop.

This function is used to dispatch events posted to a loop with no dedicated task, i.e. task name was set to NULL in `event_loop_args` argument during loop creation. This function includes an argument to limit the amount of time it runs, returning control to the caller when that time expires (or some time afterwards). There is no guarantee that a call to this function will exit at exactly the time of expiry. There is also no guarantee that events have been dispatched during the call, as the function might have spent all the allotted time waiting on the event queue. Once an event has been dequeued, however, it is guaranteed to be dispatched. This guarantee contributes to not being able to exit exactly at time of expiry as (1) blocking on internal mutexes is necessary for dispatching the dequeued event, and (2) during dispatch of the dequeued event there is no way to control the time occupied by handler code execution. The guaranteed time of exit is therefore the allotted time + amount of time required to dispatch the last dequeued event.
In cases where waiting on the queue times out, ESP_OK is returned and not ESP_ERR_TIMEOUT, since it is normal behavior.

备注: encountering an unknown event that has been posted to the loop will only generate a warning, not an error.

参数
- `event_loop` [in] event loop to dispatch posted events from, must not be NULL
- `ticks_to_run` [in] number of ticks to run the loop

返回
- ESP_OK: Success
- Others: Fail

```c
esp_err_t esp_event_handler_register(esp_event_base_t event_base, int32_t event_id, esp_event_handler_t event_handler, void *event_handler_arg)
```

Register an event handler to the system event loop (legacy).

This function can be used to register a handler for either: (1) specific events, (2) all events of a certain event base, or (3) all events known by the system event loop.

- specific events: specify exact event_base and event_id
- all events of a certain base: specify exact event_base and use ESP_EVENT_ANY_ID as the event_id
- all events known by the loop: use ESP_EVENT_ANY_BASE for event_base and ESP_EVENT_ANY_ID as the event_id

Registering multiple handlers to events is possible. Registering a single handler to multiple events is also possible. However, registering the same handler to the same event multiple times would cause the previous registrations to be overwritten.

备注: the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

参数
- `event_base` [in] the base ID of the event to register the handler for
- `event_id` [in] the ID of the event to register the handler for
- `event_handler` [in] the handler function which gets called when the event is dispatched
- `event_handler_arg` [in] data, aside from event data, that is passed to the handler when it is called

返回
- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

```c
esp_err_t esp_event_handler_register_with(esp_event_loop_handle_t event_loop, esp_event_base_t event_base, int32_t event_id, esp_event_handler_t event_handler, void *event_handler_arg)
```

Register an event handler to a specific loop (legacy).

This function behaves in the same manner as esp_event_handler_register, except the additional specification of the event loop to register the handler to.
Chapter 2. API

**备注:** the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

### 参数
- **event_loop** - [in] the event loop to register this handler function to, must not be NULL
- **event_base** - [in] the base ID of the event to register the handler for
- **event_id** - [in] the ID of the event to register the handler for
- **event_handler** - [in] the handler function which gets called when the event is dispatched
- **event_handler_arg** - [in] data, aside from event data, that is passed to the handler when it is called

### 返回
- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

```c
esp_err_t esp_event_handler_instance_register_with(esp_event_loop_handle_t event_loop,
                                                esp_event_base_t event_base, int32_t event_id,
                                                esp_event_handler_t event_handler, void *event_handler_arg,
                                                esp_event_handler_instance_t *instance)
```

Register an instance of event handler to a specific loop.

This function can be used to register a handler for either: (1) specific events, (2) all events of a certain event base, or (3) all events known by the system event loop.

- specific events: specify exact event_base and event_id
- all events of a certain base: specify exact event_base and use ESP_EVENT_ANY_ID as the event_id
- all events known by the loop: use ESP_EVENT_ANY_BASE for event_base and ESP_EVENT_ANY_ID as the event_id

Besides the error, the function returns an instance object as output parameter to identify each registration. This is necessary to remove (unregister) the registration before the event loop is deleted.

Registering multiple handlers to events, registering a single handler to multiple events as well as registering the same handler to the same event multiple times is possible. Each registration yields a distinct instance object which identifies it over the registration lifetime.

**备注:** the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

### 参数
- **event_loop** - [in] the event loop to register this handler function to, must not be NULL
- **event_base** - [in] the base ID of the event to register the handler for
- **event_id** - [in] the ID of the event to register the handler for
- **event_handler** - [in] the handler function which gets called when the event is dispatched
- **event_handler_arg** - [in] data, aside from event data, that is passed to the handler when it is called
- **instance** - [out] An event handler instance object related to the registered event handler and data, can be NULL. This needs to be kept if the specific callback instance should be unregistered before deleting the whole event loop. Registering the same event handler multiple times is possible and yields distinct instance objects. The data can be the same
for all registrations. If no unregistration is needed, but the handler should be deleted when
the event loop is deleted, instance can be NULL.

### Return
- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID or instance
  is NULL
- Others: Fail

#### esp_err_t esp_event_handler_instance_register

```c
def esp_err_t esp_event_handler_instance_register(esp_event_base_t event_base, int32_t event_id,
        esp_event_handler_t event_handler, void *event_handler_arg,
        esp_event_handler_instance_t *instance)
```

Register an instance of event handler to the default loop.

This function does the same as `esp_event_handler_instance_register_with`, except that it registers the handler
to the default event loop.

### Notes
- the event loop library does not maintain a copy of `event_handler_arg`, therefore the user should ensure
  that `event_handler_arg` still points to a valid location by the time the handler gets called

#### Parameters
- **event_base** - [in] the base ID of the event to register the handler for
- **event_id** - [in] the ID of the event to register the handler for
- **event_handler** - [in] the handler function which gets called when the event is dis-
  patched
- **event_handler_arg** - [in] data, aside from event data, that is passed to the handler
  when it is called
- **instance** - [out] An event handler instance object related to the registered event handler
  and data, can be NULL. This needs to be kept if the specific callback instance should be
  unregistered before deleting the whole event loop. Registering the same event handler
  multiple times is possible and yields distinct instance objects. The data can be the same
  for all registrations. If no unregistration is needed, but the handler should be deleted when
  the event loop is deleted, instance can be NULL.

### Return
- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID or instance
  is NULL
- Others: Fail

#### esp_err_t esp_event_handler_unregister

```c
def esp_err_t esp_event_handler_unregister(esp_event_base_t event_base, int32_t event_id,
        esp_event_handler_t event_handler)
```

Unregister a handler with the system event loop (legacy).

Unregisters a handler, so it will no longer be called during dispatch. Handlers can be unregistered for any
combination of `event_base` and `event_id` which were previously registered. To unregister a handler, the `event_base`
and `event_id` arguments must match exactly the arguments passed to `esp_event_handler_register()` when that
handler was registered. Passing `ESP_EVENT_ANY_BASE` and/or `ESP_EVENT_ANY_ID` will only unreg-
ister handlers that were registered with the same wildcard arguments.

### Notes
- When using `ESP_EVENT_ANY_ID`, handlers registered to specific event IDs using the same base
  will not be unregistered. When using `ESP_EVENT_ANY_BASE`, events registered to specific bases will also
  not be unregistered. This avoids accidental unregistration of handlers registered by other users or components.
Chapter 2. API 参考

- **event_base** - [in] the base of the event with which to unregister the handler
- **event_id** - [in] the ID of the event with which to unregister the handler
- **event_handler** - [in] the handler to unregister

返回
- ESP_OK: success
- ESP_ERR_INVALID_ARG: invalid combination of event base and event ID
- Others: fail

```c
esp_err_t esp_event_handler_unregister_with(esp_event_loop_handle_t event_loop,
                                         esp_event_base_t event_base, int32_t event_id,
                                         esp_event_handler_t event_handler)
```

Unregister a handler from a specific event loop (legacy).

This function behaves in the same manner as `esp_event_handler_unregister`, except the additional specification of the event loop to unregister the handler with.

**参数**
- **event_loop** - [in] the event loop with which to unregister this handler function, must not be NULL
- **event_base** - [in] the base of the event with which to unregister the handler
- **event_id** - [in] the ID of the event with which to unregister the handler
- **event_handler** - [in] the handler to unregister

返回
- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

```c
esp_err_t esp_event_handler_instance_unregister_with(esp_event_loop_handle_t event_loop,
                                       esp_event_base_t event_base, int32_t event_id,
                                       esp_event_handler_instance_t instance)
```

Unregister a handler instance from a specific event loop.

Unregisters a handler instance, so it will no longer be called during dispatch. Handler instances can be unregistered for any combination of event_base and event_id which were previously registered. To unregister a handler instance, the event_base and event_id arguments must match exactly the arguments passed to `esp_event_handler_instance_register()` when that handler instance was registered. Passing ESP_EVENT_ANY_BASE and/or ESP_EVENT_ANY_ID will only unregister handler instances that were registered with the same wildcard arguments.

备注: When using ESP_EVENT_ANY_ID, handlers registered to specific event IDs using the same base will not be unregistered. When using ESP_EVENT_ANY_BASE, events registered to specific bases will also not be unregistered. This avoids accidental unregistration of handlers registered by other users or components.

**参数**
- **event_loop** - [in] the event loop with which to unregister this handler function, must not be NULL
- **event_base** - [in] the base of the event with which to unregister the handler
- **event_id** - [in] the ID of the event with which to unregister the handler
- **instance** - [in] the instance object of the registration to be unregistered

返回
- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

```c
esp_err_t esp_event_handler_instance_unregister(esp_event_base_t event_base, int32_t event_id,
                                              esp_event_handler_instance_t instance)
```

Unregister a handler from the system event loop.
This function does the same as esp_event_handler_instance_unregister_with, except that it unregisters the handler instance from the default event loop.

**参数**
- `event_base` [in] the base of the event with which to unregister the handler
- `event_id` [in] the ID of the event with which to unregister the handler
- `instance` [in] the instance object of the registration to be unregistered

**返回**
- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

```c
esp_err_t esp_event_post(esp_event_base_t event_base, int32_t event_id, const void *event_data, size_t event_data_size, TickType_t ticks_to_wait)
```

Posts an event to the system default event loop. The event loop library keeps a copy of `event_data` and manages the copy’s lifetime automatically (allocation + deletion); this ensures that the data the handler receives is always valid.

**参数**
- `event_base` [in] the event base that identifies the event
- `event_id` [in] the event ID that identifies the event
- `event_data` [in] the data, specific to the event occurrence, that gets passed to the handler
- `event_data_size` [in] the size of the event data
- `ticks_to_wait` [in] number of ticks to block on a full event queue

**返回**
- ESP_OK: Success
- ESP_ERR_TIMEOUT: Time to wait for event queue to unblock expired, queue full when posting from ISR
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

```c
esp_err_t esp_event_post_to(esp_event_loop_handle_t event_loop, esp_event_base_t event_base, int32_t event_id, const void *event_data, size_t event_data_size, TickType_t ticks_to_wait)
```

Posts an event to the specified event loop. The event loop library keeps a copy of `event_data` and manages the copy’s lifetime automatically (allocation + deletion); this ensures that the data the handler receives is always valid.

This function behaves in the same manner as `esp_event_post_to`, except the additional specification of the event loop to post the event to.

**参数**
- `event_loop` [in] the event loop to post to, must not be NULL
- `event_base` [in] the event base that identifies the event
- `event_id` [in] the event ID that identifies the event
- `event_data` [in] the data, specific to the event occurrence, that gets passed to the handler
- `event_data_size` [in] the size of the event data
- `ticks_to_wait` [in] number of ticks to block on a full event queue

**返回**
- ESP_OK: Success
- ESP_ERR_TIMEOUT: Time to wait for event queue to unblock expired, queue full when posting from ISR
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID
- Others: Fail

```c
esp_err_t esp_event_isr_post(esp_event_base_t event_base, int32_t event_id, const void *event_data, size_t event_data_size, BaseType_t *task_unblocked)
```

Special variant of `esp_event_post` for posting events from interrupt handlers.
Chapter 2. API

### esp_event_isr_post_to

```c
esp_err_t esp_event_isr_post_to(esp_event_loop_handle_t event_loop, esp_event_base_t event_base, int32_t event_id, const void* event_data, size_t event_data_size, BaseType_t* task_unblocked)
```

Special variant of `esp_event_post_to` for posting events from interrupt handlers.

#### 参数
- `event_loop` [in] the event loop to post to, must not be NULL
- `event_base` [in] the event base that identifies the event
- `event_id` [in] the event ID that identifies the event
- `event_data` [in] the data, specific to the event occurrence, that gets passed to the handler
- `event_data_size` [in] the size of the event data
- `task_unblocked` [out] an optional parameter (can be NULL) which indicates that an event task with higher priority than currently running task has been unblocked by the posted event; a context switch should be requested before the interrupt is existed.

#### 返回
- ESP_OK: Success
- ESP_FAIL: Event queue for the loop full
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event ID, data size of more than 4 bytes
- Others: Fail

### esp_event_dump

```c
esp_err_t esp_event_dump(FILE* file)
```

Dumps statistics of all event loops.

Dumps event loop info in the format:
event loop
handler
handler
...
event loop
handler
handler
...

where:

```
event loop
    format: address, name rx: total_received dr: total_dropped
    where:
    address - memory address of the event loop
    name - name of the event loop, 'none' if no dedicated task
    total_received - number of successfully posted events
    total_dropped - number of events unsuccessfully posted due to queue being full
``` 

```
handler
    format: address ev: base, id inv: total_invoked run: total_runtime
    where:
    address - address of the handler function
    base, id - the event specified by event base and ID this handler executes
    total_invoked - number of times this handler has been invoked
    total_runtime - total amount of time used for invoking this handler
``` 

备注：this function is a noop when CONFIG_ESP_EVENTLOOP_PROFILING is disabled

参数 file - [in] the file stream to output to
返回

ESP_OK: Success
ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
Others: Fail

Structures

```
struct esp_event_loop_args_t
    Configuration for creating event loops.
```

Public Members

```
int32_t queue_size
    size of the event loop queue

const char *task_name
    name of the event loop task; if NULL, a dedicated task is not created for event loop

UBaseType_t task_priority
    priority of the event loop task, ignored if task name is NULL
```
uint32_t **task_stack_size**

stack size of the event loop task, ignored if task name is NULL

BaseType_t **task_core_id**

core to which the event loop task is pinned to, ignored if task name is NULL

**Header File**

- components/esp_event/include/esp_event_base.h

**Macros**

- **ESP_EVENT_DECLARE_BASE** (id)

- **ESP_EVENT_DEFINE_BASE** (id)

- **ESP_EVENT_ANY_BASE**

  register handler for any event base

- **ESP_EVENT_ANY_ID**

  register handler for any event id

**Type Definitions**

typedef void **esp_event_loop_handle_t**

a number that identifies an event with respect to a base

typedef void (**esp_event_handler_t**)(void *event_handler_arg, esp_event_base_t event_base, int32_t event_id, void *event_data)

  function called when an event is posted to the queue

typedef void **esp_event_handler_instance_t**

  context identifying an instance of a registered event handler

**Related Documents**

**2.10.10 FreeRTOS**

**Overview**

This section contains documentation of FreeRTOS types, functions, and macros. It is automatically generated from FreeRTOS header files.

**备注**: ESP-IDF FreeRTOS is based on Vanilla FreeRTOS v10.4.3

- For more information about the SMP changes of ESP-IDF FreeRTOS, see ESP-IDF FreeRTOS (SMP)
- For more information about the features added to ESP-IDF FreeRTOS, see FreeRTOS Supplemental Features.
Configuration

Vanilla FreeRTOS allows ports and applications to configure the kernel by adding various #define config... macros to FreeRTOSConfig.h. Through these macros, the kernel’s scheduling behavior and various kernel features can be enabled or disabled. However, in ESP-IDF FreeRTOS, the “FreeRTOSConfig.h“ file is considered a private and must not be modified by users. Any FreeRTOS configuration that is exposed to the user will be done so via menuconfig.

ESP-IDF FreeRTOS can be configured in the project configuration menu (idf.py menuconfig) under Component Config/FreeRTOS. The following section highlights some of the ESP-IDF FreeRTOS configuration options. For a full list of ESP-IDF FreeRTOS configurations, see Project Configuration.

- **CONFIG_FREERTOS_UNICORE** will run ESP-IDF FreeRTOS only on CPU0. Note that this is not equivalent to running Vanilla FreeRTOS. Futhermore, this option may affect behavior of components other than frees. For more details regarding the effects of running ESP-IDF FreeRTOS on a single core, refer to ESP-IDF FreeRTOS Single Core. Alternatively, users can also search for occurrences of CONFIG_FREERTOS_UNICORE in the ESP-IDF components.

- **CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER** will enclose all task functions within a wrapper function. In the case that a task function mistakenly returns (i.e. does not call vTaskDelete()), the call flow will return to the wrapper function. The wrapper function will then log an error and abort the application, as illustrated below:

```
E (25) FreeRTOS: FreeRTOS task should not return. Aborting now!
abort() was called at PC 0x40085c53 on core 0
```

ESP-IDF FreeRTOS Applications

Unlike Vanilla FreeRTOS, users must not call **vTaskStartScheduler()**. Instead, ESP-IDF FreeRTOS is started automatically. The entry point is a user defined **void app_main(void)** function.

- Typically, users would spawn the rest of their applications task from **app_main**.
- The **app_main** function is allowed to return at any point (i.e., before the application terminates).
- The **app_main** function is called from the main task.

The main task is one of multiple tasks that are automatically spawned by ESP-IDF during startup. These tasks are:

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Affinity</th>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Task (main)</td>
<td>CPU0</td>
<td>1</td>
<td>Task that simply calls <strong>app_main</strong>. This task will self delete when <strong>app_main</strong> returns</td>
</tr>
<tr>
<td>Idle Tasks (IDLEx)</td>
<td>CPU0 and CPU1</td>
<td>0</td>
<td>Idle tasks created for (and pinned to) each CPU</td>
</tr>
<tr>
<td>IPC Tasks (ipcx)</td>
<td>CPU0 and CPU1</td>
<td>24</td>
<td>IPC tasks created for (and pinned to) each CPU. IPC tasks are used to implement the IPC feature. See <strong>Inter-Processor Call</strong> for more details.</td>
</tr>
</tbody>
</table>

Task API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/task.h

Functions

BaseType_t **xTaskCreatePinnedToCore** (TaskFunction_t pxTaskCode, const char *const pcName, const configSTACK_DEPTH_TYPE usStackDepth, void *const pvParameters, UBaseType_t uxPriority, TaskHandle_t *const pvCreatedTask, const BaseType_t xCoreID)
Create a new task with a specified affinity and add it to the list of tasks that are ready to run. This function is similar to xTaskCreate, but allows setting task affinity in SMP system.

Example usage:

```c
// Task to be created.
void vTaskCode( void * pvParameters )
{
    for(;;)
    {
        // Task code goes here.
    }
}

// Function that creates a task.
void vOtherFunction( void )
{
    static uint8_t ucParameterToPass;
    TaskHandle_t xHandle = NULL;

    // Create the task pinned to core 0, storing the handle. Note that the
    // passed parameter ucParameterToPass
    // must exist for the lifetime of the task, so in this case is declared
    // static. If it was just an
    // automatic stack variable it might no longer exist, or at least have
    // been corrupted, by the time
    // the new task attempts to access it.
    xTaskCreatePinnedToCore( vTaskCode, "NAME", STACK_SIZE, &ucParameterToPass,
                             tskIDLE_PRIORITY, &xHandle, 0 );
    configASSERT( xHandle );

    // Use the handle to delete the task.
    if( xHandle != NULL )
    {
        vTaskDelete( xHandle );
    }
}
```

备注：If program uses thread local variables (ones specified with “__thread” keyword) then storage for them will be allocated on the task’s stack.

参数
- `pxTaskCode` - Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop), or should be terminated using vTaskDelete function.
- `pcName` - A descriptive name for the task. This is mainly used to facilitate debugging. Max length defined by configMAX_TASK_NAME_LEN - default is 16.
- `usStackDepth` - The size of the task stack specified as the number of bytes. Note that this differs from vanilla FreeRTOS.
- `pvParameters` - Pointer that will be used as the parameter for the task being created.
- `uxPriority` - The priority at which the task should run. Systems that include MPU support can optionally create tasks in a privileged (system) mode by setting bit portPRIVILEGE_BIT of the priority parameter. For example, to create a privileged task at priority 2 the uxPriority parameter should be set to (2 | portPRIVILEGE_BIT).
- `pvCreatedTask` - [out] Used to pass back a handle by which the created task can be referenced.
- `xCoreID` - If the value is tskNO_AFFINITY, the created task is not pinned to any CPU.
and the scheduler can run it on any core available. Values 0 or 1 indicate the index number of the CPU which the task should be pinned to. Specifying values larger than (configNUM_CORES - 1) will cause the function to fail.

Return pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h.

TaskHandle_t xTaskCreateStaticPinnedToCore(TaskFunction_t pxTaskCode, const char *const pcName, const uint32_t ulStackDepth, void *const pvParameters, UBaseType_t uxPriority, StackType_t *const pxStackBuffer, StaticTask_t *const pxTaskBuffer, const BaseType_t xCoreID)

Create a new task with a specified affinity and add it to the list of tasks that are ready to run.

This function is similar to xTaskCreateStatic, but allows specifying task affinity in an SMP system.

Example usage:

```c
#define STACK_SIZE 200

// Function that implements the task being created.
void vTaskCode( void * pvParameters )
{
    // The parameter value is expected to be 1 as 1 is passed in the
    // call to xTaskCreatePinnedToCore().
    configASSERT( ( uint32_t ) pvParameters == 1UL );

    for( ;; )
    {
        // Task code goes here.
    }
}

// Function that creates a task.
void vOtherFunction( void )
{
    TaskHandle_t xHandle = NULL;

    // Create the task pinned to core 0 without using any dynamic memory allocation.
    xHandle = xTaskCreateStaticPinnedToCore(
                 vTaskCode, "NAME", STACK_SIZE, 1, tskIDLE_PRIORITY, xStack,
                 pxTaskBuffer, 0 ); // Specify the task's core affinity
```
// puxStackBuffer and pxTaskBuffer were not NULL, so the task will have
// been created, and xHandle will be the task's handle. Use the handle
// to suspend the task.
vTaskSuspend( xHandle );
}

参数

- **pxTaskCode** – Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop), or should be terminated using vTaskDelete function.
- **pcName** – A descriptive name for the task. This is mainly used to facilitate debugging. The maximum length of the string is defined by configMAX_TASK_NAME_LEN in FreeRTOSConfig.h.
- **ulStackDepth** – The size of the task stack specified as the number of bytes. Note that this differs from vanilla FreeRTOS.
- **pvParameters** – Pointer that will be used as the parameter for the task being created.
- **uxPriority** – The priority at which the task will run.
- **pxStackBuffer** – Must point to a StackType_t array that has at least ulStackDepth indexes - the array will then be used as the task’s stack, removing the need for the stack to be allocated dynamically.
- **pxTaskBuffer** – Must point to a variable of type StaticTask_t, which will then be used to hold the task’s data structures, removing the need for the memory to be allocated dynamically.
- **xCoreID** – If the value is tskNO_AFFINITY, the created task is not pinned to any CPU, and the scheduler can run it on any core available. Values 0 or 1 indicate the index number of the CPU which the task should be pinned to. Specifying values larger than (configNUM_CORES - 1) will cause the function to fail.

返回 If neither pxStackBuffer or pxTaskBuffer are NULL, then the task will be created and pdPASS is returned. If either pxStackBuffer or pxTaskBuffer are NULL then the task will not be created and errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY is returned.

```c
static inline BaseType_t xTaskCreate( TaskFunction_t pxTaskCode, const char *const pcName, const configSTACK_DEPTH_TYPE ulStackDepth, void *const pvParameters, UBaseType_t uxPriority, TaskHandle_t * const pxCreatedTask )
```

Create a new task and add it to the list of tasks that are ready to run.

Internally, within the FreeRTOS implementation, tasks use two blocks of memory. The first block is used to hold the task’s data structures. The second block is used by the task as its stack. If a task is created using xTaskCreate() then both blocks of memory are automatically dynamically allocated inside the xTaskCreate() function. (see [https://www.FreeRTOS.org/a00111.html](https://www.FreeRTOS.org/a00111.html)). If a task is created using xTaskCreateStatic() then the application writer must provide the required memory. xTaskCreateStatic() therefore allows a task to be created without using any dynamic memory allocation.

See xTaskCreateStatic() for a version that does not use any dynamic memory allocation.

xTaskCreate() can only be used to create a task that has unrestricted access to the entire microcontroller memory map. Systems that include MPU support can alternatively create an MPU constrained task using xTaskCreateRestricted().

Example usage:

```c
// Task to be created.
void vTaskCode( void * pvParameters )
{
    for( ;; )
    {
```

(下页继续)
// Task code goes here.

// Function that creates a task.
void vOtherFunction( void )
{
    static uint8_t ucParameterToPass;
    TaskHandle_t xHandle = NULL;

    // Create the task, storing the handle. Note that the passed parameter
    ucParameterToPass
    // must exist for the lifetime of the task, so in this case is declared
    static. If it was just an
    // an automatic stack variable it might no longer exist, or at least have
    // been corrupted, by the time
    // the new task attempts to access it.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, &ucParameterToPass, tskIDLE_
    // PRIORITY, &xHandle );
    configASSERT( xHandle );

    // Use the handle to delete the task.
    if( xHandle != NULL )
    {
        vTaskDelete( xHandle );
    }
}

备注：If program uses thread local variables (ones specified with "__thread" keyword) then storage for them will be allocated on the task’s stack.

参数

- **pxTaskCode** - Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop), or should be terminated using vTaskDelete function.
- **pcName** - A descriptive name for the task. This is mainly used to facilitate debugging. Max length defined by configMAX_TASK_NAME_LEN - default is 16.
- **usStackDepth** - The size of the task stack specified as the number of bytes. Note that this differs from vanilla FreeRTOS.
- **pvParameters** - Pointer that will be used as the parameter for the task being created.
- **uxPriority** - The priority at which the task should run. Systems that include MPU support can optionally create tasks in a privileged (system) mode by setting bit portPRIVILEGE_BIT of the priority parameter. For example, to create a privileged task at priority 2 the uxPriority parameter should be set to ( 2 ^ portPRIVILEGE_BIT ).
- **pxCreatedTask** - Used to pass back a handle by which the created task can be referenced.

返回 pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

static inline TaskHandle_t xTaskCreateStatic(TaskFunction_t pxTaskCode, const char *const pcName, const uint32_t ulStackDepth, void *const pvParameters, UBaseType_t uxPriority, StackType_t *const puxStackBuffer, StaticTask_t *const pxTaskBuffer)

Create a new task and add it to the list of tasks that are ready to run.

Internally, within the FreeRTOS implementation, tasks use two blocks of memory. The first block is used to hold the task’s data structures. The second block is used by the task as its stack. If a task is created using xTaskCreate() then both blocks of memory are automatically dynamically allocated inside the xTaskCreate()
function. (see https://www.FreeRTOS.org/a00111.html). If a task is created using xTaskCreateStatic() then
the application writer must provide the required memory. xTaskCreateStatic() therefore allows a task to be
created without using any dynamic memory allocation.

Example usage:

```c
#define STACK_SIZE 200

// Dimensions the buffer that the task being created will use as its stack.
// NOTE: This is the number of bytes the stack will hold, not the number of
// words as found in vanilla FreeRTOS.

// Structure that will hold the TCB of the task being created.
StaticTask_t xTaskBuffer;

// Buffer that the task being created will use as its stack. Note this is
// an array of StackType_t variables. The size of StackType_t is dependent on
// the RTOS port.
StackType_t xStack[ STACK_SIZE ];

// Function that implements the task being created.
void vTaskCode( void * pvParameters )
{
    // The parameter value is expected to be 1 as 1 is passed in the
    // pvParameters value in the call to xTaskCreateStatic().
    configASSERT( ( uint32_t ) pvParameters == 1UL );

    // Task code goes here.
}

// Function that creates a task.
void vOtherFunction( void )
{
    TaskHandle_t xHandle = NULL;

    // Create the task without using any dynamic memory allocation.
    xHandle = xTaskCreateStatic( vTaskCode, "NAME", STACK_SIZE, ( void * ) 1, tskIDLE_PRIORITY, xStack,
                                  &xTaskBuffer ); // Variable to hold the task's data...

    // puxStackBuffer and pxTaskBuffer were not NULL, so the task will have
    // been created, and xHandle will be the task's handle. Use the handle
    // to suspend the task.
    vTaskSuspend( xHandle );
}
```

备注：If program uses thread local variables (ones specified with “__thread” keyword) then storage for them
will be allocated on the task’s stack.

参数
- `pxTaskCode` — Pointer to the task entry function. Tasks must be implemented to never
return (i.e. continuous loop), or should be terminated using vTaskDelete function.

- **pcName** - A descriptive name for the task. This is mainly used to facilitate debugging. The maximum length of the string is defined by configMAX_TASK_NAME_LEN in FreeRTOSConfig.h.

- **ulStackDepth** - The size of the stack task specified as the number of bytes. Note that this differs from vanilla FreeRTOS.

- **pvParameters** - Pointer that will be used as the parameter for the task being created.

- **uxPriority** - The priority at which the task will run.

- **puxStackBuffer** - Must point to a StackType_t array that has at least ulStackDepth indexes - the array will then be used as the task’s stack, removing the need for the stack to be allocated dynamically.

- **pxTaskBuffer** - Must point to a variable of type StaticTask_t, which will then be used to hold the task’s data structures, removing the need for the memory to be allocated dynamically.

If neither pxStackBuffer or pxTaskBuffer are NULL, then the task will be created and pdPASS is returned. If either pxStackBuffer or pxTaskBuffer are NULL then the task will not be created and errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY is returned.

```c
BaseType_t xTaskCreateRestricted(const TaskParameters_t *const pxTaskDefinition, TaskHandle_t *pxCreatedTask)
```

Only available when configSUPPORT_DYNAMIC_ALLOCATION is set to 1.

xTaskCreateRestricted() should only be used in systems that include an MPU implementation.

Create a new task and add it to the list of tasks that are ready to run. The function parameters define the memory regions and associated access permissions allocated to the task.

See xTaskCreateRestrictedStatic() for a version that does not use any dynamic memory allocation.

Example usage:

```c
// Create an TaskParameters_t structure that defines the task to be created.
static const TaskParameters_t xCheckTaskParameters = {
    vATask, // pvTaskCode - the function that implements the task.
    "ATask", // pcName - just a text name for the task to assist debugging.
    100, // usStackDepth - the stack size DEFINED IN WORDS.
    NULL, // pvParameters - passed into the task function as the function parameters.
    (1UL | portPRIVILEGE_BIT), // uxPriority - task priority, set the portPRIVILEGE_BIT if the task should run in a privileged state.
    cStackBuffer, // puxStackBuffer - the buffer to be used as the task stack.

    // xRegions - Allocate up to three separate memory regions for access by the task, with appropriate access permissions. Different processors have different memory alignment requirements - refer to the FreeRTOS documentation for full information.
    {
        // Base address Length Parameters
        { cReadWriteArray, 32, portMPU_REGION_READ_WRITE },
        { cReadOnlyArray, 32, portMPU_REGION_READ_ONLY },
        { cPrivilegedOnlyAccessArray, 128, portMPU_REGION_PRIVILEGED_READ_WRITE }
    }
};

int main( void )
{
    TaskHandle_t xHandle;
    // Add the task to the list of tasks that are ready to run.
}
```
// Create a task from the const structure defined above. The task handle
// is requested (the second parameter is not NULL) but in this case just for
// demonstration purposes as its not actually used.
xTaskCreateRestricted( &xRegTest1Parameters, &xHandle );

// Start the scheduler.
vTaskStartScheduler();

// Will only get here if there was insufficient memory to create the idle
// and/or timer task.
for(;;);

参数

• **pxTaskDefinition** – Pointer to a structure that contains a member for each of the
  normal xTaskCreate() parameters (see the xTaskCreate() API documentation) plus an
  optional stack buffer and the memory region definitions.

• **pxCreatedTask** – Used to pass back a handle by which the created task can be refer-
  enced.

返回

pdPASS if the task was successfully created and added to a ready list, otherwise an error code
defined in the file projdefs.h

```c
void vTaskAllocateMPURegions (TaskHandle_t xTask, const MemoryRegion_t *const pxRegions)
```

Only available when configSUPPORT_STATIC_ALLOCATION is set to 1.

xTaskCreateRestrictedStatic() should only be used in systems that include an MPU implementation.

Internally, within the FreeRTOS implementation, tasks use two blocks of memory. The first block is used to
hold the task’s data structures. The second block is used by the task as its stack. If a task is created using
xTaskCreateRestricted() then the stack is provided by the application writer, and the memory used to hold the
task’s data structure is automatically dynamically allocated inside the xTaskCreateRestricted() function. If a
task is created using xTaskCreateRestrictedStatic() then the application writer must provide the memory used
to hold the task’s data structures too. xTaskCreateRestrictedStatic() therefore allows a memory protected
task to be created without using any dynamic memory allocation.

Example usage:

```c
// Create an TaskParameters_t structure that defines the task to be created.
// The StaticTask_t variable is only included in the structure when
// configSUPPORT_STATIC_ALLOCATION is set to 1. The PRIVILEGED_DATA macro can
// be used to force the variable into the RTOS kernel’s privileged data area.
static PRIVILEGED_DATA StaticTask_t xTaskBuffer;
static const TaskParameters_t xCheckTaskParameters =
{
  vATask, // pvTaskCode - the function that implements the task.
  "ATask", // pcName - just a text name for the task to assist debugging.
  100,    // usStackDepth - the stack size DEFINED IN BYTES.
  NULL,   // pvParameters - passed into the task function as the function...
  --parameters.
  (1UL | portPRIVILEGE_BIT ),// uxPriority - task priority, set the
  --portPRIVILEGE_BIT if the task should run in a privileged state.
  cStackBuffer,  // puxStackBuffer - the buffer to be used as the task stack.
  // xRegions - Allocate up to three separate memory regions for access by
  // the task, with appropriate access permissions. Different processors have
  // different memory alignment requirements - refer to the FreeRTOS_
  --documentation
```
Memory regions are assigned to a restricted task when the task is created by a call to `xTaskCreateRestricted()`. These regions can be redefined using `vTaskAllocateMPURegions()`.

Example usage:

```c
// Define an array of MemoryRegion_t structures that configures an MPU region // allowing read/write access for 1024 bytes starting at the beginning of the // ucOneKByte array. The other two of the maximum 3 definable regions are // unused so set to zero.
static const MemoryRegion_t xAltRegions[ portNUM_CONFIGURABLE_REGIONS ] = {
    // Base address Length Parameters
    { ucOneKByte, 1024, portMPU_REGION_READ_WRITE },
    { 0, 0, 0 },
    { 0, 0, 0 }
};

void vATask( void *pvParameters )
{
    // This task was created such that it has access to certain regions of // memory as defined by the MPU configuration. At some point it is // desired that these MPU regions are replaced with that defined in the // xAltRegions const struct above. Use a call to vTaskAllocateMPURegions() // for this purpose. NULL is used as the task handle to indicate that this // function should modify the MPU regions of the calling task.
    vTaskAllocateMPURegions( NULL, xAltRegions );

    // Now the task can continue its function, but from this point on can only // access its stack and the ucOneKByte array (unless any other statically // defined or shared regions have been declared elsewhere).
}
```
### API Reference

#### Chapter 2. API

- **pxTaskDefinition** - Pointer to a structure that contains a member for each of the normal xTaskCreate() parameters (see the xTaskCreate() API documentation) plus an optional stack buffer and the memory region definitions. If configSUPP-ORT_STATIC_ALLOCATION is set to 1 the structure contains an additional member, which is used to point to a variable of type StaticTask_t - which is then used to hold the task’s data structure.
- **pxCreatedTask** - Used to pass back a handle by which the created task can be referenced.
- **xTask** - The handle of the task being updated.
- **pxRegions** - A pointer to a MemoryRegion_t structure that contains the new memory region definitions.

#### Return
- **pdPASS** if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h.

#### void vTaskDelete(TaskHandle_t xTaskToDelete)

INCLUDE_vTaskDelete must be defined as 1 for this function to be available. See the configuration section for more information.

Remove a task from the RTOS real time kernel’s management. The task being deleted will be removed from all ready, blocked, suspended and event lists.

**NOTE:** The idle task is responsible for freeing the kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of microcontroller processing time if your application makes any calls to vTaskDelete(). Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted.

See the demo application file death.c for sample code that utilises vTaskDelete().

**Example usage:**

```c
void vOtherFunction( void )
{
    TaskHandle_t xHandle;

    // Create the task, storing the handle.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle_--);

    // Use the handle to delete the task.
    vTaskDelete( xHandle );
}
```

- **xTaskToDelete** - The handle of the task to be deleted. Passing NULL will cause the calling task to be deleted.

#### void vTaskDelay(constTickType_t xTicksToDelay)

Delay a task for a given number of ticks. The actual time that the task remains blocked depends on the tick rate. The constant portTICK_PERIOD_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

INCLUDE_vTaskDelay must be defined as 1 for this function to be available. See the configuration section for more information.

vTaskDelay() specifies a time at which the task wishes to unblock relative to the time at which vTaskDelay() is called. For example, specifying a block period of 100 ticks will cause the task to unblock 100 ticks after vTaskDelay() is called. vTaskDelay() does not therefore provide a good method of controlling the frequency of a periodic task as the path taken through the code, as well as other task and interrupt activity, will effect the frequency at which vTaskDelay() gets called and therefore the time at which the task next executes.
xTaskDelayUntil() for an alternative API function designed to facilitate fixed frequency execution. It does this by specifying an absolute time (rather than a relative time) at which the calling task should unblock.

Example usage:

```c
void vTaskFunction( void * pvParameters )
{
    // Block for 500ms.
    const TickType_t xDelay = 500 / portTICK_PERIOD_MS;

    for( ;; )
    {
        // Simply toggle the LED every 500ms, blocking between each toggle.
        vToggleLED();
        vTaskDelay( xDelay );
    }
}
```

**参数 xTicksToDelay** —The amount of time, in tick periods, that the calling task should block.

BaseType_t xTaskDelayUntil(TickType_t *const pxPreviousWakeTime, const TickType_t xTimeIncrement)

INCLUDE xTaskDelayUntil must be defined as 1 for this function to be available. See the configuration section for more information.

Delay a task until a specified time. This function can be used by periodic tasks to ensure a constant execution frequency.

This function differs from vTaskDelay() in one important aspect: vTaskDelay() will cause a task to block for the specified number of ticks from the time vTaskDelay() is called. It is therefore difficult to use vTaskDelay() by itself to generate a fixed execution frequency as the time between a task starting to execute and that task calling vTaskDelay() may not be fixed [the task may take a different path through the code between calls, or may get interrupted or preempted a different number of times each time it executes].

Whereas vTaskDelay() specifies a wake time relative to the time at which the function is called, xTaskDelayUntil() specifies the absolute (exact) time at which it wishes to unblock.

The macro pdMS_TO_TICKS() can be used to calculate the number of ticks from a time specified in milliseconds with a resolution of one tick period.

Example usage:

```c
// Perform an action every 10 ticks.
void vTaskFunction( void * pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xFrequency = 10;
    BaseType_t xWasDelayed;

    // Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount();

    for( ;; )
    {
        // Wait for the next cycle.
        xWasDelayed = xTaskDelayUntil( &xLastWakeTime, xFrequency );

        // Perform action here. xWasDelayed value can be used to determine
        // whether a deadline was missed if the code here took too long.
    }
}
```
### 参数

- **pxPreviousWakeTime** - Pointer to a variable that holds the time at which the task was last unblocked. The variable must be initialised with the current time prior to its first use (see the example below). Following this the variable is automatically updated within xTaskDelayUntil().

- **xTimeIncrement** - The cycle time period. The task will be unblocked at time *pxPreviousWakeTime + xTimeIncrement*. Calling xTaskDelayUntil with the same xTimeIncrement parameter value will cause the task to execute with a fixed interface period.

#### 返回
Value which can be used to check whether the task was actually delayed. Will be pdTRUE if the task was delayed and pdFALSE otherwise. A task will not be delayed if the next expected wake time is in the past.

#### BaseType_t xTaskAbortDelay(TaskHandle_t xTask)

INCLUDE_xTaskAbortDelay must be defined as 1 in FreeRTOSConfig.h for this function to be available.

A task will enter the Blocked state when it is waiting for an event. The event it is waiting for can be a temporal event (waiting for a time), such as when vTaskDelay() is called, or an event on an object, such as when xQueueReceive() or ulTaskNotifyTake() is called. If the handle of a task that is in the Blocked state is used in a call to xTaskAbortDelay() then the task will leave the Blocked state, and return from whichever function call placed the task into the Blocked state.

There is no ‘FromISR’ version of this function as an interrupt would need to know which object a task was blocked on in order to know which actions to take. For example, if the task was blocked on a queue the interrupt handler would then need to know if the queue was locked.

- **参数 xTask** - The handle of the task to remove from the Blocked state.

#### 返回
If the task referenced by xTask was not in the Blocked state then pdFAIL is returned. Otherwise pdPASS is returned.

#### UBaseType_t uxTaskPriorityGet(const TaskHandle_t xTask)

INCLUDE_uxTaskPriorityGet must be defined as 1 for this function to be available. See the configuration section for more information.

Obtain the priority of any task.

#### Example usage:

```c
void vAFunction( void )
{
    TaskHandle_t xHandle;

    // Create a task, storing the handle.
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

    // ...

    // Use the handle to obtain the priority of the created task.
    // It was created with tskIDLE_PRIORITY, but may have changed
    // it itself.
    if( uxTaskPriorityGet( xHandle ) != tskIDLE_PRIORITY )
    {
        // The task has changed its priority.
    }

    // ...

    // Is our priority higher than the created task?
    if( uxTaskPriorityGet( xHandle ) < uxTaskPriorityGet( NULL ) )
    {
        // ...
    }

    // ...
}
```

(下页続き)
参数 xTask –Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.

返回 The priority of xTask.

UBaseType_t uxTaskPriorityGetFromISR (const TaskHandle_t xTask)

A version of uxTaskPriorityGet() that can be used from an ISR.

eTaskState eTaskGetState (TaskHandle_t xTask)

INCLUDE_eTaskGetState must be defined as 1 for this function to be available. See the configuration section for more information.

参数 xTask –Handle of the task to be queried.

返回 The state of xTask at the time the function was called. Note the state of the task might change between the function being called, and the functions return value being tested by the calling task.

void vTaskGetInfo (TaskHandle_t xTask, TaskStatus_t *pxTaskStatus, BaseType_t xGetFreeStackSpace, eTaskState eState)

configUSE_TRACE_FACILITY must be defined as 1 for this function to be available. See the configuration section for more information.

Populates a TaskStatus_t structure with information about a task.

Example usage:

```c
void vAFunction ( void )
{
    TaskHandle_t xHandle;
    TaskStatus_t xTaskDetails;

    // Obtain the handle of a task from its name.
    xHandle = xTaskGetHandle( "Task_Name" );

    // Check the handle is not NULL.
    configASSERT( xHandle );

    // Use the handle to obtain further information about the task.
    vTaskGetInfo( xHandle,
                  &xTaskDetails,
                  pdTRUE, // Include the high water mark in xTaskDetails.
                  eInvalid ); // Include the task state in xTaskDetails.
}
```

参数

- **xTask** –Handle of the task being queried. If xTask is NULL then information will be returned about the calling task.
- **pxTaskStatus** –A pointer to the TaskStatus_t structure that will be filled with information about the task referenced by the handle passed using the xTask parameter.
- **xGetFreeStackSpace** –The TaskStatus_t structure contains a member to report the stack high water mark of the task being queried. Calculating the stack high water mark takes a relatively long time, and can make the system temporarily unresponsive - so the xGetFreeStackSpace parameter is provided to allow the high water mark checking to be
skipped. The high watermark value will only be written to the TaskStatus_t structure if xGetFreeStackSpace is not set to pdFALSE.

- **eState** - The TaskStatus_t structure contains a member to report the state of the task being queried. Obtaining the task state is not as fast as a simple assignment - so the eState parameter is provided to allow the state information to be omitted from the TaskStatus_t structure. To obtain state information then set eState to eInvalid - otherwise the value passed in eState will be reported as the task state in the TaskStatus_t structure.

```c
void vTaskPrioritySet(TaskHandle_t xTask, UBaseType_t uxNewPriority)
```

INCLUDE_vTaskPrioritySet must be defined as 1 for this function to be available. See the configuration section for more information.

Set the priority of any task.

A context switch will occur before the function returns if the priority being set is higher than the currently executing task.

Example usage:

```c
void vAFunction(void)
{
    TaskHandle_t xHandle;

    // Create a task, storing the handle.
    xTaskCreate(vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle);

    // ...

    // Use the handle to raise the priority of the created task.
    vTaskPrioritySet(xHandle, tskIDLE_PRIORITY + 1);

    // ...

    // Use a NULL handle to raise our priority to the same value.
    vTaskPrioritySet(NULL, tskIDLE_PRIORITY + 1);
}
```

**参数**

- **xTask** - Handle to the task for which the priority is being set. Passing a NULL handle results in the priority of the calling task being set.
- **uxNewPriority** - The priority to which the task will be set.

```c
void vTaskSuspend(TaskHandle_t xTaskToSuspend)
```

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

Suspend any task. When suspended a task will never get any microcontroller processing time, no matter what its priority.

Calls to vTaskSuspend are not accumulative - i.e. calling vTaskSuspend () twice on the same task still only requires one call to vTaskResume () to ready the suspended task.

Example usage:

```c
void vAFunction(void)
{
    TaskHandle_t xHandle;
```

(下一页)
// Create a task, storing the handle.
xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle_~);

// ...

// Use the handle to suspend the created task.
vTaskSuspend( xHandle );

// ...

// The created task will not run during this period, unless
// another task calls vTaskResume( xHandle ).

//...

// Suspend ourselves.
vTaskSuspend( NULL );

// We cannot get here unless another task calls vTaskResume
// with our handle as the parameter.
}

参数 xTaskToSuspend --Handle to the task being suspended. Passing a NULL handle will
cause the calling task to be suspended.

void vTaskResume( TaskHandle_t xTaskToResume)

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section
for more information.

Resumes a suspended task.

A task that has been suspended by one or more calls to vTaskSuspend () will be made available for running
again by a single call to vTaskResume ()

Example usage:

void vAFunction( void )
{
    TaskHandle_t xHandle;

    // Create a task, storing the handle.
xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle_~);

    // ...

    // Use the handle to suspend the created task.
vTaskSuspend( xHandle );

    // ...

    // The created task will not run during this period, unless
    // another task calls vTaskResume( xHandle ).

    //...
// Resume the suspended task ourselves.
  vTaskResume( xHandle );

// The created task will once again get microcontroller processing
// time in accordance with its priority within the system.

### 参数 xTaskToResume - Handle to the task being readied.

**BaseType**: xTaskResumeFromISR (TaskHandle_t xTaskToResume)

INCLUDE_xTaskResumeFromISR must be defined as 1 for this function to be available. See the configuration section for more information.

An implementation of vTaskResume() that can be called from within an ISR.

A task that has been suspended by one or more calls to vTaskSuspend() will be made available for running again by a single call to xTaskResumeFromISR().

xTaskResumeFromISR() should not be used to synchronise a task with an interrupt if there is a chance that the interrupt could arrive prior to the task being suspended - as this can lead to interrupts being missed. Use of a semaphore as a synchronisation mechanism would avoid this eventuality.

### 参数 xTaskToResume - Handle to the task being readied.

返回 pdTRUE if resuming the task should result in a context switch, otherwise pdFALSE. This is used by the ISR to determine if a context switch may be required following the ISR.

**void vTaskStartScheduler (void)**

Starts the real time kernel tick processing. After calling the kernel has control over which tasks are executed and when.

See the demo application file main.c for an example of creating tasks and starting the kernel.

Example usage:

```c
void vAFunction( void )
{
  // Create at least one task before starting the kernel.
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );

  // Start the real time kernel with preemption.
  vTaskStartScheduler();

  // Will not get here unless a task calls vTaskEndScheduler()
}
```

备注: In ESP-IDF the scheduler is started automatically during application startup, vTaskStartScheduler() should not be called from ESP-IDF applications.

**void vTaskEndScheduler (void)**

NOTE: At the time of writing only the x86 real mode port, which runs on a PC in place of DOS, implements this function.

Stops the real time kernel tick. All created tasks will be automatically deleted and multitasking (either preemptive or cooperative) will stop. Execution then resumes from the point where vTaskStartScheduler() was called, as if vTaskStartScheduler() had just returned.

See the demo application file main.c in the demo/PC directory for an example that uses vTaskEndScheduler().
vTaskEndScheduler () requires an exit function to be defined within the portable layer (see vPortEndScheduler () in port. c for the PC port). This performs hardware specific operations such as stopping the kernel tick.

vTaskEndScheduler () will cause all of the resources allocated by the kernel to be freed - but will not free resources allocated by application tasks.

Example usage:

```c
void vTaskCode ( void * pvParameters )
{
  for (; ; )
  {
    // Task code goes here.
    // At some point we want to end the real time kernel processing
    // so call ...
    vTaskEndScheduler ();
  }
}

void vAFunction ( void )
{
  // Create at least one task before starting the kernel.
  xTaskCreate ( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
  // Start the real time kernel with preemption.
  vTaskStartScheduler ();
  // Will only get here when the vTaskCode () task has called
  // vTaskEndScheduler (). When we get here we are back to single task
  // execution.
}
```

void vTaskSuspendAll ( void )

Suspends the scheduler without disabling interrupts. Context switches will not occur while the scheduler is suspended.

After calling vTaskSuspendAll () the calling task will continue to execute without risk of being swapped out until a call to xTaskResumeAll () has been made.

API functions that have the potential to cause a context switch (for example, vTaskDelayUntil(), xQueueSend(), etc.) must not be called while the scheduler is suspended.

Example usage:

```c
void vTask1 ( void * pvParameters )
{
  for ( ; ; )
  {
    // Task code goes here.
    // ... 
    // At some point the task wants to perform a long operation during
    // which it does not want to get swapped out. It cannot use
    // taskENTER_CRITICAL ()/taskEXIT_CRITICAL () as the length of the
    // operation may cause interrupts to be missed - including the
    // ticks.
    // Prevent the real time kernel swapping out the task.
    vTaskSuspendAll ();
    // Perform the operation here. There is no need to use critical
    // sections as we have all the microcontroller processing time.
```
BaseType_t xTaskResumeAll (void)

Resumes scheduler activity after it was suspended by a call to vTaskSuspendAll().

xTaskResumeAll() only resumes the scheduler. It does not unsuspend tasks that were previously suspended by a call to vTaskSuspend().

Example usage:

```c
void vTask1 ( void *pvParameters )
{
    for (;; )
    {
        // Task code goes here.
        // ...
        // At some point the task wants to perform a long operation during
        // which it does not want to get swapped out. It cannot use
        // taskENTER_CRITICAL ()/taskEXIT_CRITICAL () as the length of the
        // operation may cause interrupts to be missed - including the
        // ticks.
        // Prevent the real time kernel swapping out the task.
        vTaskSuspendAll ();
        // Perform the operation here. There is no need to use critical
        // sections as we have all the microcontroller processing time.
        // During this time interrupts will still operate and the real
        // time kernel tick count will be maintained.
        // ...
        // The operation is complete. Restart the kernel. We want to force
        // a context switch - but there is no point if resuming the scheduler
        // caused a context switch already.
        if ( !xTaskResumeAll () )
        {
            taskYIELD ();
        }
    }
}
```

If resuming the scheduler caused a context switch then pdTRUE is returned, otherwise pdFALSE is returned.

TickType_t xTaskGetTickCount (void)

The count of ticks since vTaskStartScheduler was called.
TickType_t xTaskGetTickCountFromISR (void)

This is a version of xTaskGetTickCount() that is safe to be called from an ISR - provided that TickType_t is
the natural word size of the microcontroller being used or interrupt nesting is either not supported or not being
used.

返回 The count of ticks since vTaskStartScheduler was called.

UBaseType_t uxTaskGetNumberOfTasks (void)

返回 The number of tasks that the real time kernel is currently managing. This includes all ready,
blocked and suspended tasks. A task that has been deleted but not yet freed by the idle task
will also be included in the count.

cchar *pcTaskGetName (TaskHandle_t xTaskToQuery)

返回 The text (human readable) name of the task referenced by the handle xTaskToQuery. A task
can query its own name by either passing in its own handle, or by setting xTaskToQuery to
NULL.

TaskHandle_t xTaskGetHandle (const char* pcNameToQuery)

NOTE: This function takes a relatively long time to complete and should be used sparingly.

返回 The handle of the task that has the human readable name pcNameToQuery. NULL is re-
turned if no matching name is found. INCLUDE_xTaskGetHandle must be set to 1 in FreeR-
TOSConfig.h for pcTaskGetHandle() to be available.

UBaseType_t uxTaskGetStackHighWaterMark (TaskHandle_t xTask)

Returns the high water mark of the stack associated with xTask.

INCLUDE_uxTaskGetStackHighWaterMark must be set to 1 in FreeRTOSConfig.h for this function to be
available.

Returns the high water mark of the stack associated with xTask. That is, the minimum free stack space there
has been (in bytes not words, unlike vanilla FreeRTOS) since the task started. The smaller the returned number
the closer the task has come to overflowing its stack.

uxTaskGetStackHighWaterMark() and uxTaskGetStackHighWaterMark2() are the same except for their re-
turn type. Using configSTACK_DEPTH_TYPE allows the user to determine the return type. It gets around
the problem of the value overflowing on 8-bit types without breaking backward compatibility for applications
that expect an 8-bit return type.

参数 xTask – Handle of the task associated with the stack to be checked. Set xTask to NULL to
check the stack of the calling task.

返回 The smallest amount of free stack space there has been (in bytes not words, unlike vanilla
FreeRTOS) since the task referenced by xTask was created.

configSTACK_DEPTH_TYPE uxTaskGetStackHighWaterMark2 (TaskHandle_t xTask)

Returns the start of the stack associated with xTask.

INCLUDE_uxTaskGetStackHighWaterMark2 must be set to 1 in FreeRTOSConfig.h for this function to be
available.

Returns the high water mark of the stack associated with xTask. That is, the minimum free stack space there
has been (in bytes not words, unlike vanilla FreeRTOS) since the task started. The smaller the returned number
the closer the task has come to overflowing its stack.

uxTaskGetStackHighWaterMark() and uxTaskGetStackHighWaterMark2() are the same except for their re-
turn type. Using configSTACK_DEPTH_TYPE allows the user to determine the return type. It gets around
the problem of the value overflowing on 8-bit types without breaking backward compatibility for applications
that expect an 8-bit return type.

参数 xTask – Handle of the task associated with the stack to be checked. Set xTask to NULL to
check the stack of the calling task.

返回 The smallest amount of free stack space there has been (in bytes not words, unlike vanilla
FreeRTOS) since the task referenced by xTask was created.
uint8_t* pxTaskGetStackStart(TaskHandle_t xTask)

Returns the start of the stack associated with xTask.

INCLUDE_pxTaskGetStackStart must be set to 1 in FreeRTOSConfig.h for this function to be available.

Returns the lowest stack memory address, regardless of whether the stack grows up or down.

参数 xTask – Handle of the task associated with the stack returned. Set xTask to NULL to return the stack of the calling task.

返回 A pointer to the start of the stack.

void vTaskSetApplicationTaskTag(TaskHandle_t xTask, TaskHookFunction_t pxHookFunction)

Sets pxHookFunction to be the task hook function used by the task xTask.

参数
• xTask – Handle of the task to set the hook function for. Passing xTask as NULL has the effect of setting the calling tasks hook function.
• pxHookFunction – Pointer to the hook function.

TaskHookFunction_t xTaskGetApplicationTaskTag(TaskHandle_t xTask)

Returns the pxHookFunction value assigned to the task xTask. Do not call from an interrupt service routine - call xTaskGetApplicationTaskTagFromISR() instead.

TaskHookFunction_t xTaskGetApplicationTaskTagFromISR(TaskHandle_t xTask)

Returns the pxHookFunction value assigned to the task xTask. Can be called from an interrupt service routine.

void vTaskSetThreadLocalStoragePointer(TaskHandle_t xTaskToSet, BaseType_t xIndex, void* pvValue)

Set local storage pointer specific to the given task.

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish.

参数
• xTaskToSet – Task to set thread local storage pointer for
• xIndex – The index of the pointer to set, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.
• pvValue – Pointer value to set.

void *pvTaskGetThreadLocalStoragePointer(TaskHandle_t xTaskToQuery, BaseType_t xIndex)

Get local storage pointer specific to the given task.

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish.

参数
• xTaskToQuery – Task to get thread local storage pointer for
• xIndex – The index of the pointer to get, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.

返回 Pointer value

void vTaskSetThreadLocalStoragePointerAndDelCallback(TaskHandle_t xTaskToSet, BaseType_t xIndex, void* pvValue, TlsDeleteCallbackFunction_t pvDelCallback)

Set local storage pointer and deletion callback.

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish.

Local storage pointers set for a task can reference dynamically allocated resources. This function is similar to vTaskSetThreadLocalStoragePointer, but provides a way to release these resources when the task gets deleted.
For each pointer, a callback function can be set. This function will be called when task is deleted, with the local storage pointer index and value as arguments.

### Parameters

- **xTaskToSet** - Task to set thread local storage pointer for.
- **xIndex** - The index of the pointer to set, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.
- **pvValue** - Pointer value to set.
- **pvDelCallback** - Function to call to dispose of the local storage pointer when the task is deleted.

```c
void vApplicationGetIdleTaskMemory (StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize)
```

This function is used to provide a statically allocated block of memory to FreeRTOS to hold the Idle Task TCB. This function is required when configSUPPORT_STATIC_ALLOCATION is set. For more information see this URI: [https://www.FreeRTOS.org/a00110.html#configSUPPORT_STATIC_ALLOCATION](https://www.FreeRTOS.org/a00110.html#configSUPPORT_STATIC_ALLOCATION)

### Parameters

- **ppxIdleTaskTCBBuffer** - A handle to a statically allocated TCB buffer.
- **ppxIdleTaskStackBuffer** - A handle to a statically allocated Stack buffer for the idle task.
- **pulIdleTaskStackSize** - A pointer to the number of elements that will fit in the allocated stack buffer.

```c
BaseType_t xTaskCallApplicationTaskHook (TaskHandle_t xTask, void *pvParameter)
```

Calls the hook function associated with xTask. Passing xTask as NULL has the effect of calling the Running tasks (the calling task) hook function.

### Parameters

- **xTask** - Handle of the task to call the hook for.
- **pvParameter** - Parameter passed to the hook function for the task to interpret as it wants. The return value is the value returned by the task hook function registered by the user.

```c
TaskHandle_t xTaskGetIdleTaskHandle (void)
```

xTaskGetIdleTaskHandle() is only available if INCLUDE_xTaskGetIdleTaskHandle is set to 1 in FreeRTOSConfig.h.

Simply returns the handle of the idle task. It is not valid to call xTaskGetIdleTaskHandle() before the scheduler has been started.

```c
UBaseType_t uxTaskGetSystemState (TaskStatus_t *const pxTaskStatusArray, const UBaseType_t uxArraySize, uint32_t *const pulTotalRunTime)
```

configUSE_TRACE_FACILITY must be defined as 1 in FreeRTOSConfig.h for uxTaskGetSystemState() to be available.

uxTaskGetSystemState() populates an TaskStatus_t structure for each task in the system. TaskStatus_t structures contain, among other things, members for the task handle, task name, task priority, task state, and total amount of run time consumed by the task. See the TaskStatus_t structure definition in this file for the full member list.

### Example usage:

```c
// This example demonstrates how a human readable table of run time stats
// information is generated from raw data provided by uxTaskGetSystemState().
// The human readable table is written to pcWriteBuffer
void vTaskGetRunTimeStats ( char *pcWriteBuffer )
{
    TaskStatus_t *pxTaskStatusArray;
    volatile UBaseType_t uxArraySize, x;
}
```

(下頁続き)
uint32_t ulTotalRunTime, ulStatsAsPercentage;

// Make sure the write buffer does not contain a string.
*pcWriteBuffer = 0x00;

// Take a snapshot of the number of tasks in case it changes while this
// function is executing.
uxArraySize = uxTaskGetNumberOfTasks();

// Allocate a TaskStatus_t structure for each task. An array could be
// allocated statically at compile time.
pxTaskStatusArray = pvPortMalloc( uxArraySize * sizeof( TaskStatus_t ) );

if( pxTaskStatusArray != NULL )
{
    // Generate raw status information about each task.
    uxArraySize = uxTaskGetSystemState( pxTaskStatusArray, uxArraySize, &ulTotalRunTime );

    // For percentage calculations.
    ulTotalRunTime /= 100UL;

    // Avoid divide by zero errors.
    if( ulTotalRunTime > 0 )
    {
        // For each populated position in the pxTaskStatusArray array,
        // format the raw data as human readable ASCII data
        for( x = 0; x < uxArraySize; x++ )
        {
            // What percentage of the total run time has the task used?
            // This will always be rounded down to the nearest integer.
            // ulTotalRunTimeDiv100 has already been divided by 100.
            ulStatsAsPercentage = pxTaskStatusArray[ x ].ulRunTimeCounter / ulTotalRunTime;

            if( ulStatsAsPercentage > 0UL )
            {
                sprintf( pcWriteBuffer, "%s\t\t\t%lu\t\t%lu%%\n",-
                        pxTaskStatusArray[ x ].pcTaskName, pxTaskStatusArray[ x ].ulRunTimeCounter,-
                        ulStatsAsPercentage );
            }
            else
            {
                // If the percentage is zero here then the task has
                // consumed less than 1% of the total run time.
                sprintf( pcWriteBuffer, "%s\t\t\t<1%%\n",-
                        pxTaskStatusArray[ x ].pcTaskName, pxTaskStatusArray[ x ].ulRunTimeCounter );
            }
            pcWriteBuffer += strlen( ( char * ) pcWriteBuffer );
        }
    }
    else
    {
        // The array is no longer needed, free the memory it consumes.
        vPortFree( pxTaskStatusArray );
    }
}
参数

- **pxTaskStatusArray** – A pointer to an array of TaskStatus_t structures. The array must contain at least one TaskStatus_t structure for each task that is under the control of the RTOS. The number of tasks under the control of the RTOS can be determined using the uxTaskGetNumberOfTasks() API function.
- **uxArraySize** – The size of the array pointed to by the pxTaskStatusArray parameter. The size is specified as the number of indexes in the array, or the number of TaskStatus_t structures contained in the array, not by the number of bytes in the array.
- **pulTotalRunTime** – If configGENERATE_RUN_TIME_STATS is set to 1 in FreeRTOSConfig.h then *pulTotalRunTime is set by uxTaskGetSystemState() to the total run time (as defined by the run time stats clock, see https://www.FreeRTOS.org/rtos-run-time-stats.html) since the target booted. pulTotalRunTime can be set to NULL to omit the total run time information.

返回

The number of TaskStatus_t structures that were populated by uxTaskGetSystemState(). This should equal the number returned by the uxTaskGetNumberOfTasks() API function, but will be zero if the value passed in the uxArraySize parameter was too small.

```c
void vTaskList (char *pcWriteBuffer)

List all the current tasks.

configUSE_TRACE_FACILITY and configUSE_STATS_FORMATTING_FUNCTIONS must both be defined as 1 for this function to be available. See the configuration section of the FreeRTOS.org website for more information.

NOTE 1: This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

Lists all the current tasks, along with their current state and stack usage high water mark.

Tasks are reported as blocked (‘B’), ready (‘R’), deleted (‘D’) or suspended (‘S’).

PLEASE NOTE:

This function is provided for convenience only, and is used by many of the demo applications. Do not consider it to be part of the scheduler.

vTaskList() calls uxTaskGetSystemState(), then formats part of the uxTaskGetSystemState() output into a human readable table that displays task names, states and stack usage.

vTaskList() has a dependency on the sprintf() C library function that might bloat the code size, use a lot of stack, and provide different results on different platforms. An alternative, tiny, third party, and limited functionality implementation of sprintf() is provided in many of the FreeRTOS/Demo sub-directories in a file called printf-stdarg.c (note printf-stdarg.c does not provide a full sprintf() implementation!).

It is recommended that production systems call uxTaskGetSystemState() directly to get access to raw stats data, rather than indirectly through a call to vTaskList().

参数 **pcWriteBuffer** – A buffer into which the above mentioned details will be written, in ASCII form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.

```
configured by the portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() macro. Calling vTaskGetRunTimeStats() writes the total execution time of each task into a buffer, both as an absolute count value and as a percentage of the total system execution time.

NOTE 2:
This function is provided for convenience only, and is used by many of the demo applications. Do not consider it to be part of the scheduler.

vTaskGetRunTimeStats() calls uxTaskGetSystemState(), then formats part of the uxTaskGetSystemState() output into a human readable table that displays the amount of time each task has spent in the Running state in both absolute and percentage terms.

vTaskGetRunTimeStats() has a dependency on the sprintf() C library function that might bloat the code size, use a lot of stack, and provide different results on different platforms. An alternative, tiny, third party, and limited functionality implementation of sprintf() is provided in many of the FreeRTOS/Demo sub-directories in a file called printf-stdarg.c (note printf-stdarg.c does not provide a full snprintf() implementation!).

It is recommended that production systems call uxTaskGetSystemState() directly to get access to raw stats data, rather than indirectly through a call to vTaskGetRunTimeStats().

参数

pcWriteBuffer - A buffer into which the execution times will be written, in ASCII form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.

uint32_t ulTaskGetIdleRunTimeCounter (void)

configGENERATE_RUN_TIME_STATS and configUSE_STATS_FORMATTING_FUNCTIONS must both be defined as 1 for this function to be available. The application must also then provide definitions for portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() and portGET_RUN_TIME_COUNTER_VALUE() to configure a peripheral timer/counter and return the timer’s current count value respectively. The counter should be at least 10 times the frequency of the tick count.

Setting configGENERATE_RUN_TIME_STATS to 1 will result in a total accumulated execution time being stored for each task. The resolution of the accumulated time value depends on the frequency of the timer configured by the portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() macro. While uxTaskGetSystemState() and vTaskGetRunTimeStats() writes the total execution time of each task into a buffer, ulTaskGetIdleRunTimeCounter() returns the total execution time of just the idle task.

返回
The total run time of the idle task. This is the amount of time the idle task has actually been executing. The unit of time is dependent on the frequency configured using the portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() and portGET_RUN_TIME_COUNTER_VALUE() macros.

BaseType_t xTaskGenericNotify (TaskHandle_t xTaskToNotify, UBaseType_t uxIndexToNotify, uint32_t ulValue, eNotifyAction eAction, uint32_t *pulPreviousNotificationValue)


configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these functions to be available.

Sends a direct to task notification to a task, with an optional value and action.

Each task has a private array of “notification values” (or ‘notifications’), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task’s notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.
A task can use xTaskNotifyWaitIndexed() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTakeIndexed() to [optionally] block to wait for a notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWaitIndexed() or ulTaskNotifyTakeIndexed() (or their un-indexed equivalents). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

**NOTE** Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotify() is the original API function, and remains backward compatible by always operating on the notification value at index 0 in the array. Calling xTaskNotify() is equivalent to calling xTaskNotifyIndexed() with the uxIndexToNotify parameter set to 0.

- **eSetBits** - The target notification value is bitwise ORed with ulValue. xTaskNotifyIndexed() always returns pdPASS in this case.
- **eIncrement** - The target notification value is incremented. ulValue is not used and xTaskNotifyIndexed() always returns pdPASS in this case.
- **eSetValueWithOverwrite** - The target notification value is set to the value of ulValue, even if the task being notified had not yet processed the previous notification at the same array index (the task already had a notification pending at that index). xTaskNotifyIndexed() always returns pdPASS in this case.
- **eSetValueWithoutOverwrite** - If the task being notified did not already have a notification pending at the same array index then the target notification value is set to ulValue and xTaskNotifyIndexed() will return pdPASS. If the task being notified already had a notification pending at the same array index then no action is performed and pdFAIL is returned.
- **eNoAction** - The task receives a notification at the specified array index without the notification value at that index being updated. ulValue is not used and xTaskNotifyIndexed() always returns pdPASS in this case.

**参数**
- **xTaskToNotify** – The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- **uxIndexToNotify** – The index within the target task’s array of notification values to which the notification is to be sent. uxIndexToNotify must be less than config-TASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotify() does not have this parameter and always sends notifications to index 0.
- **ulValue** – Data that can be sent with the notification. How the data is used depends on the value of the eAction parameter.
- **eAction** – Specifies how the notification updates the task’s notification value, if at all. Valid values for eAction are as follows:
  - **pulPreviousNotificationValue** – Can be used to pass out the subject task’s notification value before any bits are modified by the notify function.

**返回**
Dependent on the value of eAction. See the description of the eAction parameter.

```c
BaseType_t xTaskGenericNotifyFromISR(TaskHandle_t xTaskToNotify, UBaseType_t uxIndexToNotify, uint32_t ulValue, eNotifyAction eAction, uint32_t *pulPreviousNotificationValue, BaseType_t *pxHigherPriorityTaskWoken)
```


configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these functions to be available.

A version of xTaskNotifyIndexed() that can be used from an interrupt service routine (ISR).
Each task has a private array of “notification values” (or ‘notifications’), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task’s notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A task can use xTaskNotifyWaitIndexed() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTakeIndexed() to [optionally] block to wait for a notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWaitIndexed() or ulTaskNotifyTakeIndexed() (or their un-indexed equivalents). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyFromISR() is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling xTaskNotifyFromISR() is equivalent to calling xTaskNotifyIndexedFromISR() with the uxIndexToNotify parameter set to 0.

eSetBits - The task’s notification value is bitwise ORed with ulValue. xTaskNotify() always returns pdPASS in this case.

eIncrement - The task’s notification value is incremented. ulValue is not used and xTaskNotify() always returns pdPASS in this case.

eSetValueWithOverwrite - The task’s notification value is set to the value of ulValue, even if the task being notified had not yet processed the previous notification (the task already had a notification pending). xTaskNotify() always returns pdPASS in this case.

eSetValueWithoutOverwrite - If the task being notified did not already have a notification pending then the task’s notification value is set to ulValue and xTaskNotify() will return pdPASS. If the task being notified already had a notification pending then no action is performed and pdFAIL is returned.

eNoAction - The task receives a notification without its notification value being updated. ulValue is not used and xTaskNotify() always returns pdPASS in this case.

参数
- **uxIndexToNotify** – The index within the target task’s array of notification values to which the notification is to be sent. uxIndexToNotify must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyFromISR() does not have this parameter and always sends notifications to index 0.
- **xTaskToNotify** – The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- **ulValue** – Data that can be sent with the notification. How the data is used depends on the value of the eAction parameter.
- **eAction** – Specifies how the notification updates the task’s notification value, if at all. Valid values for eAction are as follows:

  - **eSetBits** – The task’s notification value is bitwise ORed with ulValue.
  - **eIncrement** – The task’s notification value is incremented. ulValue is not used.
  - **eSetValueWithOverwrite** – The task’s notification value is set to ulValue.
  - **eSetValueWithoutOverwrite** – If the task already had a notification pending then no action is performed.
  - **eNoAction** – The task receives a notification without its notification value being updated. ulValue is not used.

 submit document feedback
• **pxHigherPriorityTaskWoken** - xTaskNotifyFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending the notification caused the task to which the notification was sent to leave the Blocked state, and the unblocked task has a priority higher than the currently running task. If xTaskNotifyFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited. How a context switch is requested from an ISR is dependent on the port - see the documentation page for the port in use.

返回 Dependent on the value of eAction. See the description of the eAction parameter.

```c
BaseType_t xTaskGenericNotifyWait( UBaseType_t uxIndexToWorldOn, uint32_t ulBitsToClearOnEntry, uint32_t ulBitsToClearOnExit, uint32_t *pulNotificationValue, TickType_t xTicksToWait)
```
Waits for a direct to task notification to be pending at a given index within an array of direct to task notifications. See [https://www.FreeRTOS.org/RTOS-task-notifications.html](https://www.FreeRTOS.org/RTOS-task-notifications.html) for details.

**configUSE_TASK_NOTIFICATIONS** must be undefined or defined as 1 for this function to be available.

Each task has a private array of “notification values” (or ‘notifications’), each of which is a 32-bit unsigned integer (uint32_t). The constant **configTASK_NOTIFICATION_ARRAY_ENTRIES** sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task’s notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWaitIndexed() or ulTaskNotifyTake Indexed() (or their un-indexed equivalents). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

A task can use xTaskNotifyWaitIndexed() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTake Indexed() to [optionally] block to wait for a notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

**NOTE** Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyWait() is the original API function, and remains backward compatible by always operating on the notification value at index 0 in the array. Calling xTaskNotifyWait() is equivalent to calling xTaskNotifyWaitIndexed() with the uxIndexToWorldOn parameter set to 0.

**参数**

- **uxIndexToWorldOn** – The index within the calling task’s array of notification values on which the calling task will wait for a notification to be received. uxIndexToWorldOn must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyWait() does not have this parameter and always waits for notifications on index 0.

- **ulBitsToClearOnEntry** – Bits that are set in ulBitsToClearOnEntry value will be cleared in the calling task’s notification value before the task is marked as waiting for a new notification (provided a notification is not already pending). Optionally blocks if no notifications are pending. Setting ulBitsToClearOnEntry to ULONG_MAX (if limits.h is included) or OxffffffUL (if limits.h is not included) will have the effect of resetting the task’s notification value to 0. Setting ulBitsToClearOnEntry to 0 will leave the task’s notification value unchanged.
• **ulBitsToClearOnExit** - If a notification is pending or received before the calling task exits the `xTaskNotifyWait()` function then the task’s notification value (see the `xTaskNotify()` API function) is passed out using the `pulNotificationValue` parameter. Then any bits that are set in `ulBitsToClearOnExit` will be cleared in the task’s notification value (note * `pulNotificationValue` is set before any bits are cleared). Setting `ulBitsToClearOnExit` to `ULONG_MAX` (if `limits.h` is included) or `0xffffffffUL` (if `limits.h` is not included) will have the effect of resetting the task’s notification value to 0 before the function exits. Setting `ulBitsToClearOnExit` to 0 will leave the task’s notification value unchanged when the function exits (in which case the value passed out in `pulNotificationValue` will match the task’s notification value).

• **pulNotificationValue** – Used to pass the task’s notification value out of the function. Note the value passed out will not be effected by the clearing of any bits caused by `ulBitsToClearOnExit` being non-zero.

• **xTicksToWait** - The maximum amount of time that the task should wait in the Blocked state for a notification to be received, should a notification not already be pending when `xTaskNotifyWait()` was called. The task will not consume any processing time while it is in the Blocked state. This is specified in kernel ticks, the macro `pdMS_TO_TICKS(value_in_ms)` can be used to convert a time specified in milliseconds to a time specified in ticks.

If a notification was received (including notifications that were already pending when `xTaskNotifyWait()` was called) then `pdPASS` is returned. Otherwise `pdFAIL` is returned.

```c
void vTaskGenericNotifyGiveFromISR(TaskHandle_t xTaskToNotify, UBaseType_t uxIndexToNotify, BaseType_t *pxHigherPriorityTaskWoken)
```

A version of `xTaskNotifyGiveIndexed()` that can be called from an interrupt service routine (ISR).


`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for this macro to be available.

Each task has a private array of “notification values” (or “notifications”), each of which is a 32-bit unsigned integer (`uint32_t`). The constant `configTASK_NOTIFICATION_ARRAY_ENTRIES` sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task’s notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

`vTaskNotifyGiveIndexedFromISR()` is intended for use when task notifications are used as light weight and faster binary or counting semaphore equivalents. Actual FreeRTOS semaphores are given from an ISR using the `xSemaphoreGiveFromISR()` API function, the equivalent action that instead uses a task notification is `vTaskNotifyGiveIndexedFromISR()`.

When task notifications are being used as a binary or counting semaphore equivalent then the task being notified should wait for the notification using the `ulTaskNotificationTakeIndexed()` API function rather than the `xTaskNotifyWaitIndexed()` API function.

**NOTE** Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. `xTaskNotifyFromISR()` is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling `xTaskNotifyGiveFromISR()` is equivalent to calling `xTaskNotifyGiveIndexedFromISR()` with the `uxIndexToNotify` parameter set to 0.
**xTaskToNotify** - The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().

**uxIndexToNotify** - The index within the target task’s array of notification values to which the notification is to be sent. uxIndexToNotify must be less than configTASK_NOTIFICATION_ARRAY_ENTRIES. xTaskNotifyGiveFromISR() does not have this parameter and always sends notifications to index 0.

**pxHigherPriorityTaskWoken** - vTaskNotifyGiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending the notification caused the task to which the notification was sent to leave the Blocked state, and the unblocked task has a priority higher than the currently running task. If vTaskNotifyGiveFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited. How a context switch is requested from an ISR is dependent on the port - see the documentation page for the port in use.

```
uint32_t ulTaskGenericNotifyTake( UBaseType_t uxIndexToWaitOn, BaseType_t xClearCountOnExit, 
    TickType_t xTicksToWait)
```

Waits for a direct to task notification on a particular index in the calling task’s notification array in a manner similar to taking a counting semaphore.


configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this function to be available.

Each task has a private array of “notification values” (or ‘notifications’), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task’s notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

ulTaskNotifyTakeIndexed() is intended for use when a task notification is used as a faster and lighter weight binary or counting semaphore alternative. Actual FreeRTOS semaphores are taken using the xSemaphoreTake() API function, the equivalent action that instead uses a task notification is ulTaskNotifyTakeIndexed().

When a task is using its notification value as a binary or counting semaphore other tasks should send notifications to it using the xTaskNotifyGiveIndexed() macro, or xTaskNotifyIndex() function with the eAction parameter set to eIncrement.

ulTaskNotifyTakeIndexed() can either clear the task’s notification value at the array index specified by the uxIndexToWaitOn parameter to zero on exit, in which case the notification value acts like a binary semaphore, or decrement the notification value on exit, in which case the notification value acts like a counting semaphore.

A task can use ulTaskNotifyTakeIndexed() to [optionally] block to wait for the task’s notification value to be non-zero. The task does not consume any CPU time while it is in the Blocked state.

Where as xTaskNotifyWaitIndexed() will return when a notification is pending, ulTaskNotifyTakeIndexed() will return when the task’s notification value is not zero.

**NOTE** Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. ulTaskNotifyTake() is the original API function, and remains backward compatible by always operating on the notification value at index 0 in the array. Calling ulTaskNotifyTake() is equivalent to calling ulTaskNotifyTakeIndexed() with the uxIndexToWaitOn parameter set to 0.
参数

- **uxIndexToWaitOn** – The index within the calling task’s array of notification values on which the calling task will wait for a notification to be non-zero. `uxIndexToWaitOn` must be less than `configTASK_NOTIFICATION_ARRAY_ENTRIES`. `xFnTaskNotifyTake()` does not have this parameter and always waits for notifications on index 0.

- **xClearCountOnExit** – if `xClearCountOnExit` is `pdFALSE` then the task’s notification value is decremented when the function exits. In this way the notification value acts like a counting semaphore. If `xClearCountOnExit` is not `pdFALSE` then the task’s notification value is cleared to zero when the function exits. In this way the notification value acts like a binary semaphore.

- **xTicksToWait** – The maximum amount of time that the task should wait in the Blocked state for the task’s notification value to be greater than zero, should the count not already be greater than zero when `uTaskNotifyTake()` was called. The task will not consume any processing time while it is in the Blocked state. This is specified in kernel ticks, the macro `pdMS_TO_TICKS(value_in_ms)` can be used to convert a time specified in milliseconds to a time specified in ticks.

返回 The task’s notification count before it is either cleared to zero or decremented (see the `xClearCountOnExit` parameter).

```c
BaseType_t xTaskGenericNotifyStateClear (TaskHandle_t xTask, UBaseType_t uxIndexToClear)
```


`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for these functions to be available.

Each task has a private array of “notification values” (or ‘notifications’), each of which is a 32-bit unsigned integer (`uint32_t`). The constant `configTASK_NOTIFICATION_ARRAY_ENTRIES` sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

If a notification is sent to an index within the array of notifications then the notification at that index is said to be ‘pending’ until it is read or explicitly cleared by the receiving task. `xTaskNotifyStateClearIndexed()` is the function that clears a pending notification without reading the notification value. The notification value at the same array index is not altered. Set `xTask` to `NULL` to clear the notification state of the calling task.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. `xTaskNotifyStateClear()` is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling `xTaskNotifyStateClear()` is equivalent to calling `xTaskNotifyStateClearIndexed()` with the `uxIndexToNotify` parameter set to 0.

参数

- **xTask** – The handle of the RTOS task that will have a notification state cleared. Set `xTask` to `NULL` to clear a notification state in the calling task. To obtain a task’s handle create the task using `xTaskCreate()` and make use of the `pxCreateTask` parameter, or create the task using `xTaskCreateStatic()` and store the returned value, or use the task’s name in a call to `xTaskGetHandle()`.

- **uxIndexToClear** – The index within the target task’s array of notification values to act upon. For example, setting `uxIndexToClear` to 1 will clear the state of the notification at index 1 within the array. `uxIndexToClear` must be less than `configTASK_NOTIFICATION_ARRAY_ENTRIES`. `uTaskNotifyStateClear()` does not have this parameter and always acts on the notification at index 0.

返回 `pdTRUE` if the task’s notification state was set to `eNotWaitingNotification`, otherwise `pdFALSE`.

```c
uint32_t ulTaskGenericNotifyValueClear (TaskHandle_t xTask, UBaseType_t uxIndexToClear, uint32_t ulBitsToClear)
```


`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for these functions to be available.
Each task has a private array of “notification values” (or ‘notifications’), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK-notification_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

ulTaskNotifyValueClearIndexed() clears the bits specified by the ulBitsToClear bit mask in the notification value at array index uxIndexToClear of the task referenced by xTask.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value”, and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. ulTaskNotifyValueClear() is the original API function, and remains backward compatible by always operating on the notification value at index 0 within the array. Calling ulTaskNotifyValueClear() is equivalent to calling ulTaskNotifyValueClearIndexed() with the uxIndexToClear parameter set to 0.

### 参数
- **xTask** – The handle of the RTOS task that will have bits in one of its notification values cleared. Set xTask to NULL to clear bits in a notification value of the calling task. To obtain a task’s handle create the task using xTaskCreate() and make use of the pxCreatedTask parameter, or create the task using xTaskCreateStatic() and store the returned value, or use the task’s name in a call to xTaskGetHandle().
- **uxIndexToClear** – The index within the target task’s array of notification values in which to clear the bits. uxIndexToClear must be less than configTASK-notification_ARRAY_ENTRIES. ulTaskNotifyValueClear() does not have this parameter and always clears bits in the notification value at index 0.
- **ulBitsToClear** – Bitmask of the bits to clear in the notification value of xTask. Set a bit to 1 to clear the corresponding bits in the task’s notification value. Set ulBitsToClear to 0xffffffff (UINT_MAX on 32-bit architectures) to clear the notification value to 0. Set ulBitsToClear to 0 to query the task’s notification value without clearing any bits.

### 返回
The value of the target task’s notification value before the bits specified by ulBitsToClear were cleared.

```c
void vTaskSetTimeOutState ( TimeOut_t *const pxTimeOut )
```

```c
BaseType_t xTaskCheckForTimeOut ( TimeOut_t *const pxTimeOut, TickType_t *const pxTicksToWait )
```

Determines if pxTicksToWait ticks has passed since a time was captured using a call to vTaskSetTimeOutState(). The captured time includes the tick count and the number of times the tick count has overflowed.

### Example Usage:

```c
// Driver library function used to receive uxWantedBytes from an Rx buffer
// that is filled by a UART interrupt. If there are not enough bytes in the
// Rx buffer then the task enters the Blocked state until it is notified that
// more data has been placed into the buffer. If there is still not enough
// data then the task re-enters the Blocked state, and xTaskCheckForTimeOut()
// is used to re-calculate the Block time to ensure the total amount of time
// spent in the Blocked state does not exceed MAX_TIME_TO_WAIT. This
// continues until either the buffer contains at least uxWantedBytes bytes,
// or the total amount of time spent in the Blocked state reaches
// MAX_TIME_TO_WAIT - at which point the task reads however many bytes are
// available up to a maximum of uxWantedBytes.

size_t xUART_Receive( uint8_t *pucBuffer, size_t uxWantedBytes )
{
    size_t uxReceived = 0;
    TickType_t xTicksToWait = MAX_TIME_TO_WAIT;
    TimeOut_t xTimeOut;

    // Initialize xTimeOut. This records the time at which this function
    // was entered.
```
vTaskSetTimeOutState( &xTimeOut );

// Loop until the buffer contains the wanted number of bytes, or a // timeout occurs.
while( UART_bytes_in_rx_buffer( pxUARTInstance ) < uxWantedBytes )
{
  // The buffer didn't contain enough data so this task is going to // enter the Blocked state. Adjusting xTicksToWait to account for // any time that has been spent in the Blocked state within this // function so far to ensure the total amount of time spent in the // Blocked state does not exceed MAX_TIME_TO_WAIT.
  if( xTaskCheckForTimeOut( &xTimeOut, &xTicksToWait ) != pdFALSE )
  {
    // Timed out before the wanted number of bytes were available, // exit the loop.
    break;
  }

  // Wait for a maximum of xTicksToWait ticks to be notified that the // receive interrupt has placed more data into the buffer.
  ulTaskNotifyTake( pdTRUE, xTicksToWait );
}

// Attempt to read uxWantedBytes from the receive buffer into pucBuffer. // The actual number of bytes read (which might be less than // uxWantedBytes) is returned.
uxReceived = UART_read_from_receive_buffer( pxUARTInstance, pucBuffer, uxWantedBytes );

return uxReceived;

参见:
https://www.FreeRTOS.org/xTaskCheckForTimeOut.html

参数
- pxTimeOut – The time status as captured previously using vTaskSetTimeOutState. If the timeout has not yet occurred, it is updated to reflect the current time status.
- pxTicksToWait – The number of ticks to check for timeout i.e. if pxTicksToWait ticks have passed since pxTimeOut was last updated (either by vTaskSetTimeOutState() or xTaskCheckForTimeOut()), the timeout has occurred. If the timeout has not occurred, pxTicksToWait is updated to reflect the number of remaining ticks.

返回 若timeout has occurred, pdTRUE is returned. Otherwise pdFALSE is returned and pxTicksToWait is updated to reflect the number of remaining ticks.

BaseType_t xTaskCatchUpTicks (TickType_t xTicksToCatchUp)

Macros

tskKERNEL_VERSION_NUMBER

tskKERNEL_VERSION_MAJOR

tskKERNEL_VERSION_MINOR

tskKERNEL_VERSION_BUILD
tskMPU_REGION_READ_ONLY

tskMPU_REGION_READ_WRITE

tskMPU_REGION_EXECUTE_NEVER

tskMPU_REGION_NORMAL_MEMORY

tskMPU_REGION_DEVICE_MEMORY

tskDEFAULT_INDEX_TO_NOTIFY

tskNO_AFFINITY

tskIDLE_PRIORITY

Defines the priority used by the idle task. This must not be modified.

taskYIELD()

Macro for forcing a context switch.

taskENTER_CRITICAL(x)

Macro to mark the start of a critical code region. Preemptive context switches cannot occur when in a critical region.

备注: This may alter the stack (depending on the portable implementation) so must be used with care!

taskENTER_CRITICAL_FROM_ISR()

taskENTER_CRITICAL_ISR(x)

taskEXIT_CRITICAL(x)

Macro to mark the end of a critical code region. Preemptive context switches cannot occur when in a critical region.

备注: This may alter the stack (depending on the portable implementation) so must be used with care!

taskEXIT_CRITICAL_FROM_ISR(x)

taskEXIT_CRITICAL_ISR(x)

taskDISABLE_INTERRUPTS()

Macro to disable all maskable interrupts.

taskENABLE_INTERRUPTS()

Macro to enable microcontroller interrupts.

taskSCHEDULER_SUSPENDED

taskSCHEDULER_NOT_STARTED

taskSCHEDULER_RUNNING
vTaskDelayUntil (pxPreviousWakeTime, xTimeIncrement)

xTaskNotify (xTaskToNotify, ulValue, eAction)

xTaskNotifyIndexed (xTaskToNotify, ulValue, eAction)

xTaskNotifyAndQuery (xTaskToNotify, ulValue, eAction, pulPreviousNotifyValue)


xTaskNotifyAndQueryIndexed() performs the same operation as xTaskNotifyIndexed() with the addition that it also returns the subject task’s prior notification value (the notification value at the time the function is called rather than when the function returns) in the additional pulPreviousNotifyValue parameter.

xTaskNotifyAndQuery() performs the same operation as xTaskNotify() with the addition that it also returns the subject task’s prior notification value (the notification value as it was at the time the function is called, rather than when the function returns) in the additional pulPreviousNotifyValue parameter.

xTaskNotifyAndQueryIndexed (xTaskToNotify, ulValue, eAction, pulPreviousNotifyValue)

xTaskNotifyFromISR (xTaskToNotify, ulValue, eAction, pxHigherPriorityTaskWoken)

xTaskNotifyIndexedFromISR (xTaskToNotify, ulValue, eAction, pxHigherPriorityTaskWoken)

xTaskNotifyAndQueryIndexedFromISR (xTaskToNotify, ulValue, eAction, pulPreviousNotificationValue, pxHigherPriorityTaskWoken)


xTaskNotifyAndQueryIndexedFromISR() performs the same operation as xTaskNotifyIndexedFromISR() with the addition that it also returns the subject task’s prior notification value (the notification value at the time the function is called rather than at the time the function returns) in the additional pulPreviousNotificationValue parameter.

xTaskNotifyAndQueryFromISR() performs the same operation as xTaskNotifyFromISR() with the addition that it also returns the subject task’s prior notification value (the notification value at the time the function is called rather than at the time the function returns) in the additional pulPreviousNotificationValue parameter.

xTaskNotifyAndQueryFromISR (xTaskToNotify, ulValue, eAction, pulPreviousNotificationValue, pxHigherPriorityTaskWoken)

xTaskNotifyWait (ulBitsToClearOnEntry, ulBitsToClearOnExit, pulNotificationValue, xTicksToWait)

xTaskNotifyWaitIndexed (ulBitsToClearOnEntry, ulBitsToClearOnExit, ulValue, eAction, xTicksToWait)

xTaskNotifyGiveIndexed (xTaskToNotify, ulValue, eAction)

Sends a direct to task notification to a particular index in the target task’s notification array in a manner similar to giving a counting semaphore.


configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for these macros to be available.

Each task has a private array of “notification values” (or “notifications”), each of which is a 32-bit unsigned integer (uint32_t). The constant configTASK_NOTIFICATION_ARRAY_ENTRIES sets the number of indexes in the array, and (for backward compatibility) defaults to 1 if left undefined. Prior to FreeRTOS V10.4.0 there was only one notification value per task.

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.
A notification sent to a task can optionally perform an action, such as update, overwrite or increment one of the task’s notification values. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

xTaskNotifyGiveIndexed() is a helper macro intended for use when task notifications are used as light weight and faster binary or counting semaphore equivalents. Actual FreeRTOS semaphores are given using the xSemaphoreGive() API function, the equivalent action that instead uses a task notification is xTaskNotifyGiveIndexed().

When task notifications are being used as a binary or counting semaphore equivalent then the task being notified should wait for the notification using the ulTaskNotificationTakeIndexed() API function rather than the xTaskNotifyWaitIndexed() API function.

NOTE Each notification within the array operates independently - a task can only block on one notification within the array at a time and will not be unblocked by a notification sent to any other array index.

Backward compatibility information: Prior to FreeRTOS V10.4.0 each task had a single “notification value” , and all task notification API functions operated on that value. Replacing the single notification value with an array of notification values necessitated a new set of API functions that could address specific notifications within the array. xTaskNotifyGive() is the original API function, and remains backward compatible by always operating on the notification value at index 0 in the array. Calling xTaskNotifyGive() is equivalent to calling xTaskNotifyGiveIndexed() with the uxIndexToNotify parameter set to 0.

参数
- xTaskToNotify – The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- uxIndexToNotify – The index within the target task’s array of notification values to which the notification is to be sent. uxIndexToNotify must be less than config-TASK-notification ARRAY_ENTRIES. xTaskNotifyGive() does not have this parameter and always sends notifications to index 0.

返回 xTaskNotifyGive() is a macro that calls xTaskNotify() with the eAction parameter set to eIncrement - so pdPASS is always returned.

**Type Definitions**

typedef struct tskTaskControlBlock *TaskHandle_t

typedef BaseType_t (*TaskHookFunction_t)(void*)

typedef void (*TlsDeleteCallbackFunction_t)(int, void*)

Prototype of local storage pointer deletion callback.
Chapter 2. API 参考

Enumerations

enum eTaskState
    Task states returned by eTaskGetState.
    Values:
        enumerator eRunning
        enumerator eReady
        enumerator eBlocked
        enumerator eSuspended
        enumerator eDeleted
        enumerator eInvalid

enum eNotifyAction
    Values:
        enumerator eNoAction
        enumerator eSetBits
        enumerator eIncrement
        enumerator eSetValueWithOverwrite
        enumerator eSetValueWithoutOverwrite

enum eSleepModeStatus
    Possible return values for cTaskConfirmSleepModeStatus().
    Values:
        enumerator eAbortSleep
        enumerator eStandardSleep
        enumerator eNoTasksWaitingTimeout

Queue API

Header File

- components/freertos/FreeRTOS-Kernel/include/freertos/queue.h
Functions

`BaseType_t xQueueGenericSend(QueueHandle_t xQueue, const void *pvItemToQueue, TickType_t xTicksToWait, const BaseType_t xCopyPosition)`

It is preferred that the macros `xQueueSend()`, `xQueueSendToFront()` and `xQueueSendToBack()` are used in place of calling this function directly.

Post an item on a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See `xQueueSendFromISR()` for an alternative which may be used in an ISR.

Example usage:

```c
struct AMessage
{
    char ucMessageID;
    char ucData[20];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
    QueueHandle_t xQueue1, xQueue2;
    struct AMessage *pxMessage;

    // Create a queue capable of containing 10 uint32_t values.
    xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMessage ) );

    // ...
    if( xQueue1 != 0 )
    {
        // Send an uint32_t. Wait for 10 ticks for space to become
        // available if necessary.
        if( xQueueGenericSend( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10, /*.queueSEND_TO_BACK */ ) != pdPASS )
        {
            // Failed to post the message, even after 10 ticks.
        }
    }

    if( xQueue2 != 0 )
    {
        // Send a pointer to a struct AMessage object. Don't block if the
        // queue is already full.
        pxMessage = &xMessage;
        xQueueGenericSend( xQueue2, ( void * ) pxMessage, ( TickType_t ) 0, /*.queueSEND_TO_BACK */ );
    }

    // ... Rest of task code.
}
```

参数

- `xQueue` - The handle to the queue on which the item is to be posted.
- `pvItemToQueue` - A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from `pvItemToQueue` into the queue storage area.
• `xTicksToWait` - The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant `portTICK_PERIOD_MS` should be used to convert to real time if this is required.
• `xCopyPosition` - Can take the value `queueSEND_TO_BACK` to place the item at the back of the queue, or `queueSEND_TO_FRONT` to place the item at the front of the queue (for high priority messages).

```c
_mode(pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

BaseType_t xQueuePeek (QueueHandle_t xQueue, void *const pvBuffer, TickType_t xTicksToWait)

Receive an item from a queue without removing the item from the queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to `xQueueReceive()`.

This macro must not be used in an interrupt service routine. See `xQueuePeekFromISR()` for an alternative that can be called from an interrupt service routine.

Example usage:
```c
struct AMessage
{
    char ucMessageID;
    char ucData[20];
} xMessage;
QueueHandle_t xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
    struct AMessage *pxMessage;

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
    if( xQueue == 0 )
    {
        // Failed to create the queue.
    }
    ...

    // Send a pointer to a struct AMessage object. Don't block if the queue is already full.
    pxMessage = & xMessage;
    xQueueSend( xQueue, ( void * ) &pxMessage, ( TickType_t ) 0 );

    // ... Rest of task code.
}

// Task to peek the data from the queue.
void vADifferentTask( void *pvParameters )
{
    struct AMessage *pxRxedMessage;

    if( xQueue != 0 )
    {
        // Peek a message on the created queue. Block for 10 ticks if a
```
Chapter 2. API

// message is not immediately available.
if (xQueuePeek( xQueue, & pxRxedMessage ), ( TickType_t ) 10 )
{
    // pcRxedMessage now points to the struct AMessage variable posted
    // by vATask, but the item still remains on the queue.
}

// ... Rest of task code.

参数
- **xQueue** - The handle to the queue from which the item is to be received.
- **pvBuffer** - Pointer to the buffer into which the received item will be copied.
- **xTicksToWait** - The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required. xQueuePeek() will return immediately if xTicksToWait is 0 and the queue is empty.

返回 pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

```c
BaseType_t xQueuePeekFromISR (QueueHandle_t xQueue, void*const pvBuffer)
```

A version of xQueuePeek() that can be called from an interrupt service routine (ISR).

Receive an item from a queue without removing the item from the queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to xQueueReceive().

参数
- **xQueue** - The handle to the queue from which the item is to be received.
- **pvBuffer** - Pointer to the buffer into which the received item will be copied.

返回 pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

```c
BaseType_t xQueueReceive (QueueHandle_t xQueue, void*const pvBuffer, TickType_t xTicksToWait)
```

Receive an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items are removed from the queue.

This function must not be used in an interrupt service routine. See xQueueReceiveFromISR for an alternative that can.

Example usage:

```c
struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

QueueHandle_t xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
    struct AMessage *pxMessage;
```
```c
// Create a queue capable of containing 10 pointers to AMessage structures. // These should be passed by pointer as they contain a lot of data.
xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
if( xQueue == 0 )
{
    // Failed to create the queue.
}

// Send a pointer to a struct AMessage object. Don't block if the // queue is already full.
pxMessage = & xMessage;
xQueueSend( xQueue, ( void * ) &pxMessage, ( TickType_t ) 0 );

// ... Rest of task code.

// Task to receive from the queue.
void vADifferentTask( void * pvParameters )
{
    struct AMessage * pxRxedMessage;
    if( xQueue != 0 )
    {
        // Receive a message on the created queue. Block for 10 ticks if a // message is not immediately available.
        if( xQueueReceive( xQueue, & pxRxedMessage , ( TickType_t ) 10 ) )
        {
            // pcRxedMessage now points to the struct AMessage variable posted // by vATask.
        }
    }

    // ... Rest of task code.
}
```

参数

- `xQueue` - The handle to the queue from which the item is to be received.
- `pvBuffer` - Pointer to the buffer into which the received item will be copied.
- `xTicksToWait` - The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. `xQueueReceive()` will return immediately if `xTicksToWait` is zero and the queue is empty. The time is defined in tick periods so the constant `portTICK_PERIOD_MS` should be used to convert to real time if this is required.

返回 `pdTRUE` if an item was successfully received from the queue, otherwise `pdFALSE`.

```c
UBaseType_t uxQueueMessagesWaiting( const QueueHandle_t xQueue )
Return the number of messages stored in a queue.
```

参数 `xQueue` - A handle to the queue being queried.

```c
UBaseType_t uxQueueSpacesAvailable( const QueueHandle_t xQueue )
Return the number of free spaces available in a queue. This is equal to the number of items that can be sent to the queue before the queue becomes full if no items are removed.
```

参数 `xQueue` - A handle to the queue being queried.
void **QueueDelete**(QueueHandle_t xQueue)

Delete a queue - freeing all the memory allocated for storing of items placed on the queue.

参数 xQueue - A handle to the queue to be deleted.

BaseType_t **QueueGenericSendFromISR**(QueueHandle_t xQueue, const void *const pvItemToQueue, BaseType_t *const pxHigherPriorityTaskWoken, const BaseType_t xCopyPosition)

It is preferred that the macros **QueueSendFromISR()**, **QueueSendToFrontFromISR()** and **QueueSendToBackFromISR()** be used in place of calling this function directly. **QueueGiveFromISR()** is an equivalent for use by semaphores that don’t actually copy any data.

Post an item on a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```c
void vBufferISR( void )
{
    char cIn;
    BaseType_t xHigherPriorityTaskWokenByPost;

    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWokenByPost = pdFALSE;

    // Loop until the buffer is empty.
    do {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

        // Post each byte.
        xQueueGenericSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWokenByPost, queueSEND_TO_BACK );
    } while( portINPUT_BYTE( BUFFER_COUNT ) );

    // Now the buffer is empty we can switch context if necessary. Note that the
    // name of the yield function required is port specific.
    if( xHigherPriorityTaskWokenByPost )
    {
        taskYIELD_YIELD_FROM_ISR();
    }
}
```

参数

- **xQueue** - The handle to the queue on which the item is to be posted.
- **pvItemToQueue** - A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** [out] **QueueGenericSendFromISR()** will set *pxHigherPriorityTaskWoken* to *pdTRUE* if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If **QueueGenericSendFromISR()** sets this value to *pdTRUE* then a context switch should be requested before the interrupt is exited.
- **xCopyPosition** - Can take the value queueSEND_TO_BACK to place the item at the back of the queue, or queueSEND_TO_FRONT to place the item at the front of the queue (for high priority messages).

返回 *pdTRUE* if the data was successfully sent to the queue, otherwise errQUEUE_FULL.
BaseType_t xQueueGiveFromISR(QueueHandle_t xQueue, BaseType_t *const pxHigherPriorityTaskWoken)

BaseType_t xQueueReceiveFromISR(QueueHandle_t xQueue, void *const pvBuffer, BaseType_t *const pxHigherPriorityTaskWoken)

Receive an item from a queue. It is safe to use this function from within an interrupt service routine.

Example usage:

```c
QueueHandle_t xQueue;

// Function to create a queue and post some values.
void vAFunction( void *pvParameters )
{
    char cValueToPost;
    const TickType_t xTicksToWait = (TickType_t)0xff;

    // Create a queue capable of containing 10 characters.
    xQueue = xQueueCreate( 10, sizeof( char ) );
    if( xQueue == 0 )
    {
        // Failed to create the queue.
    }
    // ...

    // Post some characters that will be used within an ISR. If the queue
    // is full then this task will block for xTicksToWait ticks.
    cValueToPost = 'a';
    xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );
    cValueToPost = 'b';
    xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );

    // ... keep posting characters ... this task may block when the queue
    // becomes full.
    cValueToPost = 'c';
    xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );
}

// ISR that outputs all the characters received on the queue.
void vISR_Routine( void )
{
    BaseType_t xTaskWokenByReceive = pdFALSE;
    char cRxedChar;

    while( xQueueReceiveFromISR( xQueue, ( void * ) &cRxedChar, & xTaskWokenByReceive ) )
    {
        // A character was received. Output the character now.
        vOutputCharacter( cRxedChar );

        // If removing the character from the queue woke the task that was
        // posting onto the queue cTaskWokenByReceive will have been set to
        // pdTRUE. No matter how many times this loop iterates only one
        // task will be woken.
    }
    if( cTaskWokenByPost != ( char ) pdFALSE;
    {
        taskYIELD();
    }
```

(下页继续)
参数
- **xQueue** - The handle to the queue from which the item is to be received.
- **pvBuffer** - Pointer to the buffer into which the received item will be copied.
- **pxHigherPriorityTaskWoken** - [out] A task may be blocked waiting for space to become available on the queue. If xQueueReceiveFromISR causes such a task to unblock, *pxTaskWoken will get set to pdTRUE, otherwise **pxTaskWoken will remain unchanged.

返回 pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

```c
BaseType_t xQueueIsQueueEmptyFromISR(const QueueHandle_t xQueue)

BaseType_t xQueueIsQueueFullFromISR(const QueueHandle_t xQueue)

UBaseType_t uxQueueMessagesWaitingFromISR(const QueueHandle_t xQueue)

void vQueueAddToRegistry(QueueHandle_t xQueue, const char* pcQueueName)
```

The registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call vQueueAddToRegistry() add a queue, semaphore or mutex handle to the registry if you want the handle to be available to a kernel aware debugger. If you are not using a kernel aware debugger then this function can be ignored.

```c
configQUEUE_REGISTRY_SIZE defines the maximum number of handles the registry can hold. configQUEUE_REGISTRY_SIZE must be greater than 0 within FreeRTOSConfig.h for the registry to be available. Its value does not effect the number of queues, semaphores and mutexes that can be created - just the number that the registry can hold.
```

参数
- **xQueue** - The handle of the queue being added to the registry. This is the handle returned by a call to xQueueCreate(). Semaphore and mutex handles can also be passed in here.
- **pcQueueName** - The name to be associated with the handle. This is the name that the kernel aware debugger will display. The queue registry only stores a pointer to the string - so the string must be persistent (global or preferably in ROM/Flash), not on the stack.

```c
void vQueueUnregisterQueue(QueueHandle_t xQueue)
```

The registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call vQueueAddToRegistry() add a queue, semaphore or mutex handle to the registry if you want the handle to be available to a kernel aware debugger, and vQueueUnregisterQueue() to remove the queue, semaphore or mutex from the register. If you are not using a kernel aware debugger then this function can be ignored.

参数 **xQueue** - The handle of the queue being removed from the registry.

```c
const char* pcQueueGetName(QueueHandle_t xQueue)
```

The queue registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call pcQueueGetName() to look up and return the name of a queue in the queue registry from the queue’s handle.

参数 **xQueue** - The handle of the queue the name of which will be returned.

返回 If the queue is in the registry then a pointer to the name of the queue is returned. If the queue is not in the registry then NULL is returned.

```c
QueueHandle_t xQueueGenericCreate(const UBaseType_t uxQueueLength, const UBaseType_t uxItemSize, const uint8_t ucQueueType)
```

Generic version of the function used to create a queue using dynamic memory allocation. This is called by other functions and macros that create other RTOS objects that use the queue structure as their base.

```c
QueueHandle_t xQueueGenericCreateStatic(const UBaseType_t uxQueueLength, const UBaseType_t uxItemSize, uint8_t* pucQueueStorage, StaticQueue_t* pxStaticQueue, const uint8_t ucQueueType)
```
Generic version of the function used to create a queue using dynamic memory allocation. This is called by other functions and macros that create other RTOS objects that use the queue structure as their base.

**QueueSetHandle_t xQueueCreateSet**(const UBaseType_t uxEventQueueLength)

Queue sets provide a mechanism to allow a task to block (pend) on a read operation from multiple queues or semaphores simultaneously.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

A queue set must be explicitly created using a call to xQueueCreateSet() before it can be used. Once created, standard FreeRTOS queues and semaphores can be added to the set using calls to xQueueAddToSet(). xQueueSelectFromSet() is then used to determine which, if any, of the queues or semaphores contained in the set is in a state where a queue read or semaphore take operation would be successful.

Note 1: See the documentation on [https://www.FreeRTOS.org/RTOS-queue-sets.html](https://www.FreeRTOS.org/RTOS-queue-sets.html) for reasons why queue sets are very rarely needed in practice as there are simpler methods of blocking on multiple objects.

Note 2: Blocking on a queue set that contains a mutex will not cause the mutex holder to inherit the priority of the blocked task.

Note 3: An additional 4 bytes of RAM is required for each space in a every queue added to a queue set. Therefore counting semaphores that have a high maximum count value should not be added to a queue set.

Note 4: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

参数 **uxEventQueueLength** — Queue sets store events that occur on the queues and semaphores contained in the set. uxEventQueueLength specifies the maximum number of events that can be queued at once. To be absolutely certain that events are not lost uxEventQueueLength should be set to the total sum of the length of the queues added to the set, where binary semaphores and mutexes have a length of 1, and counting semaphores have a length set by their maximum count value. Examples:

- If a queue set is to hold a queue of length 5, another queue of length 12, and a binary semaphore, then uxEventQueueLength should be set to \((5 + 12 + 1)\), or 18.
- If a queue set is to hold three binary semaphores then uxEventQueueLength should be set to \((1 + 1 + 1)\), or 3.
- If a queue set is to hold a counting semaphore that has a maximum count of 5, and a counting semaphore that has a maximum count of 3, then uxEventQueueLength should be set to \((5 + 3)\), or 8.

返回 If the queue set is created successfully then a handle to the created queue set is returned. Otherwise NULL is returned.

**BaseType_t xQueueAddToSet**(QueueSetMemberHandle_t xQueueOrSemaphore, QueueSetHandle_t xQueueSet)

Adds a queue or semaphore to a queue set that was previously created by a call to xQueueCreateSet().

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

Note 1: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

参数

- **xQueueOrSemaphore** — The handle of the queue or semaphore being added to the queue set (cast to an QueueSetMemberHandle_t type).
- **xQueueSet** — The handle of the queue set to which the queue or semaphore is being added.

返回 If the queue or semaphore was successfully added to the queue set then pdPASS is returned.
If the queue could not be successfully added to the queue set because it is already a member of a different queue set then pdFAIL is returned.
**Chapter 2. API**

### BaseType_t xQueueRemoveFromSet

*(QueueSetMemberHandle_t xQueueOrSemaphore, QueueSetHandle_t xQueueSet)*

Removes a queue or semaphore from a queue set. A queue or semaphore can only be removed from a set if the queue or semaphore is empty.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

**参数**
- **xQueueOrSemaphore** – The handle of the queue or semaphore being removed from the queue set (cast to a QueueSetMemberHandle_t type).
- **xQueueSet** – The handle of the queue set in which the queue or semaphore is included.

**返回** If the queue or semaphore was successfully removed from the queue set then pdPASS is returned. If the queue was not in the queue set, or the queue (or semaphore) was not empty, then pdFAIL is returned.

### QueueSetMemberHandle_t xQueueSelectFromSet

*(QueueSetHandle_t xQueueSet, const TickType_t xTicksToWait)*

xQueueSelectFromSet() selects from the members of a queue set a queue or semaphore that either contains data (in the case of a queue) or is available to take (in the case of a semaphore). xQueueSelectFromSet() effectively allows a task to block (pend) on a read operation on all the queues and semaphores in a queue set simultaneously.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

**Note 1:** See the documentation on [https://www.FreeRTOS.org/RTOS-queue-sets.html](https://www.FreeRTOS.org/RTOS-queue-sets.html) for reasons why queue sets are very rarely needed in practice as there are simpler methods of blocking on multiple objects.

**Note 2:** Blocking on a queue set that contains a mutex will not cause the mutex holder to inherit the priority of the blocked task.

**Note 3:** A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

**参数**
- **xQueueSet** – The queue set on which the task will (potentially) block.
- **xTicksToWait** – The maximum time, in ticks, that the calling task will remain in the Blocked state (with other tasks executing) to wait for a member of the queue set to be ready for a successful queue read or semaphore take operation.

**返回** xQueueSelectFromSet() will return the handle of a queue (cast to a QueueSetMemberHandle_t type) contained in the queue set that contains data, or the handle of a semaphore (cast to a QueueSetMemberHandle_t type) contained in the queue set that is available, or NULL if no such queue or semaphore exists before before the specified block time expires.

### QueueSetMemberHandle_t xQueueSelectFromSetFromISR

*(QueueSetHandle_t xQueueSet)*

A version of xQueueSelectFromSet() that can be used from an ISR.

### Macros

**xQueueCreate**(uxQueueLength, uxItemSize)

Creates a new queue instance, and returns a handle by which the new queue can be referenced.

Internally, within the FreeRTOS implementation, queues use two blocks of memory. The first block is used to hold the queue’s data structures. The second block is used to hold items placed into the queue. If a queue is created using xQueueCreate() then both blocks of memory are automatically dynamically allocated inside the xQueueCreate() function. (see [https://www.FreeRTOS.org/a00111.html](https://www.FreeRTOS.org/a00111.html)). If a queue is created using xQueueCreateStatic() then the application writer must provide the memory that will get used by the queue. xQueueCreateStatic() therefore allows a queue to be created without using any dynamic memory allocation.


**Example usage:**
struct AMessage
{
    char ucMessageID;
    char ucData[20];
};

void vATask( void *pvParameters )
{
    QueueHandle_t xQueue1, xQueue2;

    // Create a queue capable of containing 10 uint32_t values.
    xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );
    if( xQueue1 == 0 )
    {
        // Queue was not created and must not be used.
    }

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );
    if( xQueue2 == 0 )
    {
        // Queue was not created and must not be used.
    }

    // ... Rest of task code.
}

ьюQueueLength – The maximum number of items that the queue can contain.
ьюItemSize – The number of bytes each item in the queue will require. Items are
queued by copy, not by reference, so this is the number of bytes that will be copied for
each posted item. Each item on the queue must be the same size.

If the queue is successfully create then a handle to the newly created queue is returned. If
the queue cannot be created then 0 is returned.

xQueueCreateStatic(ьюQueueLength,ьюItemSize, pucQueueStorage, pxQueueBuffer)

Creates a new queue instance, and returns a handle by which the new queue can be referenced.

Internally, within the FreeRTOS implementation, queues use two blocks of memory. The first block is used
to hold the queue’s data structures. The second block is used to hold items placed into the queue. If a queue
is created using xQueueCreate() then both blocks of memory are automatically dynamically allocated inside
the xQueueCreate() function. (see https://www.FreeRTOS.org/a00111.html). If a queue is created using
xQueueCreateStatic() then the application writer must provide the memory that will get used by the queue.
xQueueCreateStatic() therefore allows a queue to be created without using any dynamic memory allocation.


Example usage:

struct AMessage
{
    char ucMessageID;
    char ucData[20];
};

#define QUEUE_LENGTH 10
#define ITEM_SIZE sizeof( uint32_t )

// xQueueBuffer will hold the queue structure.
StaticQueue_t xQueueBuffer;

// ucQueueStorage will hold the items posted to the queue. Must be at least
// (queue length) * (queue item size) bytes long.
uint8_t ucQueueStorage[ QUEUE_LENGTH * ITEM_SIZE ];

void vATask( void *pvParameters )
{
    QueueHandle_t xQueue;

    // Create a queue capable of containing 10 uint32_t values.
    xQueue = xQueueCreate( QUEUE_LENGTH, // The number of items the queue can...
                        ITEM_SIZE   // The size of each item in the queue
                        & ucQueueStorage[ 0 ] ), // The buffer that will...
                        ITEM_SIZE   // The size of each item in the queue
                        & xQueueBuffer ); // The buffer that will hold the...
                        ITEM_SIZE   // The size of each item in the queue

    // The queue is guaranteed to be created successfully as no dynamic memory
    // allocation is used. Therefore xQueue is now a handle to a valid queue.
    // ... Rest of task code.
}

参数

• uxQueueLength – The maximum number of items that the queue can contain.
• uxItemSize – The number of bytes each item in the queue will require. Items are
    queued by copy, not by reference, so this is the number of bytes that will be copied for
    each posted item. Each item on the queue must be the same size.
• pucQueueStorage – If uxItemSize is not zero then pucQueueStorageBuffer must point
    to a uint8_t array that is at least large enough to hold the maximum number of items that
    can be in the queue at any one time - which is ( uxQueueLength * uxItemsSize ) bytes. If
    uxItemSize is zero then pucQueueStorageBuffer can be NULL.
• pxQueueBuffer – Must point to a variable of type StaticQueue_t, which will be used
    to hold the queue’s data structure.

返回 If the queue is created then a handle to the created queue is returned. If pxQueueBuffer is
NULL then NULL is returned.

xQueueSendToFront (xQueue, pvItemToQueue, xTicksToWait)
Post an item to the front of a queue. The item is queued by copy, not by reference. This function must not be
called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in
an ISR.

Example usage:

```c
struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
    QueueHandle_t xQueue1, xQueue2;
```
struct AMessage *pxMessage;

// Create a queue capable of containing 10 uint32_t values.
xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

// ...
if( xQueue1 != 0 )
{
    // Send an uint32_t. Wait for 10 ticks for space to become
    // available if necessary.
    if( xQueueSendToFront( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) != pdPASS )
    {
        // Failed to post the message, even after 10 ticks.
    }
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object. Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSendToFront( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
}

// ... Rest of task code.

### 参数
- **xQueue** - The handle to the queue on which the item is to be posted.
- **pvItemToQueue** - A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **xTicksToWait** - The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

### 返回
- **pdTRUE** if the item was successfully posted, otherwise **errQUEUE_FULL**.

**xQueueSendToBack**(xQueue, pvItemToQueue, xTicksToWait)

This is a macro that calls xQueueGenericSend().

Post an item to the back of a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR() for an alternative which may be used in an ISR.

Example usage:

```c
struct AMessage
{
    char ucMessageID;
    char ucData[ 20 ];
} xMessage;
```
void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

    // Create a queue capable of containing 10 uint32_t values.
    xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

    ...

if( xQueue1 != 0 )
{
    // Send an uint32_t. Wait for 10 ticks for space to become
    // available if necessary.
    if( xQueueSendToBack( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) !-pdPASS )
    {
        // Failed to post the message, even after 10 ticks.
    }
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object. Don't block if the
    // queue is already full.
    pxMessage = &xMessage;
    xQueueSendToBack( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
}

    // ... Rest of task code.
}

参数
• xQueue  - The handle to the queue on which the item is to be posted.
• pvItemToQueue - A pointer to the item that is to be placed on the queue. The size of
  the items the queue will hold was defined when the queue was created, so this many bytes
  will be copied from pvItemToQueue into the queue storage area.
• xTicksToWait - The maximum amount of time the task should block waiting for space
  to become available on the queue, should it already be full. The call will return immediately
  if this is set to 0 and the queue is full. The time is defined in tick periods so the constant
  portTICK_PERIOD_MS should be used to convert to real time if this is required.

返回  pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

xQueueSend(xQueue, pvItemToQueue, xTicksToWait)
This is a macro that calls xQueueGenericSend(). It is included for backward compatibility with versions of
FreeRTOS.org that did not include the xQueueSendToFront() and xQueueSendToBack() macros. It is equiva-
lent to xQueueSendToBack().

Post an item on a queue. The item is queued by copy, not by reference. This function must not be called from
an interrupt service routine. See xQueueSendFromISR() for an alternative which may be used in an ISR.

Example usage:
struct AMessage
{
    char ucMessageID;
    char ucData[20];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
    QueueHandle_t xQueue1, xQueue2;
    struct AMessage *pxMessage;

    // Create a queue capable of containing 10 uint32_t values.
    xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

    // Create a queue capable of containing 10 pointers to AMessage structures.
    // These should be passed by pointer as they contain a lot of data.
    xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

    // ...
    if( xQueue1 != 0 )
    {
        // Send an uint32_t. Wait for 10 ticks for space to become
        // available if necessary.
        if( xQueueSend( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) != pdPASS )
        {
            // Failed to post the message, even after 10 ticks.
        }
    }

    if( xQueue2 != 0 )
    {
        // Send a pointer to a struct AMessage object. Don't block if the
        // queue is already full.
        pxMessage = & xMessage;
        xQueueSend( xQueue2, ( void * ) pxMessage, ( TickType_t ) 0 );
    }

    // ... Rest of task code.
}

**参数**

- xQueue - The handle to the queue on which the item is to be posted.
- pvItemToQueue - A pointer to the item that is to be placed on the queue. The size of
  the items the queue will hold was defined when the queue was created, so this many bytes
  will be copied from pvItemToQueue into the queue storage area.
- xTicksToWait - The maximum amount of time the task should block waiting for space
  to become available on the queue, should it already be full. The call will return immediately
  if this is set to 0 and the queue is full. The time is defined in tick periods so the constant
  portTICK_PERIOD_MS should be used to convert to real time if this is required.

**返回**

pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

xQueueOverwrite (xQueue, pvItemToQueue)

Only for use with queues that have a length of one - so the queue is either empty or full.

Post an item on a queue. If the queue is already full then overwrite the value held in the queue. The item is
queued by copy, not by reference.
This function must not be called from an interrupt service routine. See xQueueOverwriteFromISR() for an alternative which may be used in an ISR.

Example usage:

```c
void vFunction( void *pvParameters )
{
    QueueHandle_t xQueue;
    uint32_t ulVarToSend, ulValReceived;

    // Create a queue to hold one uint32_t value. It is strongly recommended *not* to use xQueueOverwrite() on queues that can contain more than one value, and doing so will trigger an assertion if configASSERT() is defined.
    xQueue = xQueueCreate( 1, sizeof( uint32_t ) );

    // Write the value 10 to the queue using xQueueOverwrite().
    ulVarToSend = 10;
    xQueueOverwrite( xQueue, &ulVarToSend );

    // Peeking the queue should now return 10, but leave the value 10 in the queue. A block time of zero is used as it is known that the queue holds a value.
    ulValReceived = 0;
    xQueuePeek( xQueue, &ulValReceived, 0 );

    if( ulValReceived != 10 )
    {
        // Error unless the item was removed by a different task.
    }

    // The queue is still full. Use xQueueOverwrite() to overwrite the value held in the queue with 100.
    ulVarToSend = 100;
    xQueueOverwrite( xQueue, &ulVarToSend );

    // This time read from the queue, leaving the queue empty once more. A block time of 0 is used again.
    xQueueReceive( xQueue, &ulValReceived, 0 );

    // The value read should be the last value written, even though the queue was already full when the value was written.
    if( ulValReceived != 100 )
    {
        // Error!
    }

    // ...
}
```

**Params**

- `xQueue` - The handle of the queue to which the data is being sent.
- `pvItemToQueue` - A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.

**Return**

xQueueOverwrite() is a macro that calls xQueueGenericSend(), and therefore has the same return values as xQueueSendToFront(). However, pdPASS is the only value that can be returned because xQueueOverwrite() will write to the queue even when the queue is already full.
xQueueSendToFrontFromISR (xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR().

Post an item to the front of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```c
void vBufferISR ( void )
{
    char cIn;
    BaseType_t xHigherPriorityTaskWoken;

    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWoken = pdFALSE;

    // Loop until the buffer is empty.
    do
    {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

        // Post the byte.
        xQueueSendToFrontFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );
    } while ( portINPUT_BYTE( BUFFER_COUNT ) );

    // Now the buffer is empty we can switch context if necessary.
    if( xHigherPriorityTaskWoken )
    {
        portYIELD_FROM_ISR ( );
    }
}
```

参数

- **xQueue** – The handle to the queue on which the item is to be posted.
- **pvItemToQueue** – A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** – [out] xQueueSendToFrontFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to un-block, and the unblocked task has a priority higher than the currently running task. If xQueueSendToBackFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

xQueueSendToBackFromISR (xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR().

Post an item to the back of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):
```c
void vBufferISR( void )
{
    char cIn;
    BaseType_t xHigherPriorityTaskWoken;

    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWoken = pdFALSE;

    // Loop until the buffer is empty.
    do
    {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );
        // Post the byte.
        xQueueSendToBackFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );
    } while ( portINPUT_BYTE( BUFFER_COUNT ) );

    // Now the buffer is empty we can switch context if necessary.
    if( xHigherPriorityTaskWoken )
    {
        portYIELD_FROM_ISR( );
    }
}
```

### 参数
- `xQueue` - The handle to the queue on which the item is to be posted.
- `pvItemToQueue` - A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `pxHigherPriorityTaskWoken` - [out] `xQueueSendToBackFromISR()` will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If `xQueueSendToBackFromISR()` sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

### 返回
- pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

### xQueueOverwriteFromISR(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)
A version of `xQueueOverwrite()` that can be used in an interrupt service routine (ISR).

Only for use with queues that can hold a single item - so the queue is either empty or full.

Post an item on a queue. If the queue is already full then overwrite the value held in the queue. The item is queued by copy, not by reference.

Example usage:

```c
QueueHandle_t xQueue;

void vFunction( void *pvParameters )
{
    // Create a queue to hold one uint32_t value. It is strongly recommended not to use xQueueOverwriteFromISR() on queues that can contain more than one value, and doing so will trigger an assertion if configASSERT() is defined.
    xQueue = xQueueCreate( 1, sizeof( uint32_t ) );
}
```
```c
void vAnInterruptHandler( void )
{
    // xHigherPriorityTaskWoken must be set to pdFALSE before it is used.
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    uint32_t ulVarToSend, ulValReceived;

    // Write the value 10 to the queue using xQueueOverwriteFromISR().
    ulVarToSend = 10;
    xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );

    // The queue is full, but calling xQueueOverwriteFromISR() again will still
    // pass because the value held in the queue will be overwritten with the
    // new value.
    ulVarToSend = 100;
    xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );

    // Reading from the queue will now return 100.
    // ...

    if( xHigherPriorityTaskWoken == pdTRUE )
    {
        // Writing to the queue caused a task to unblock and the unblocked task
        // has a priority higher than or equal to the priority of the currently
        // executing task (the task this interrupt interrupted). Perform a...
        // switch so this interrupt returns directly to the unblocked task.
        portYIELD_FROM_ISR(); // or portEND_SWITCHING_ISR() depending on the port.
    }
}
```

### 参数
- **xQueue** - The handle to the queue on which the item is to be posted.
- **pvItemToQueue** - A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** - [out] xQueueOverwriteFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueOverwriteFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

### 返回
xQueueOverwriteFromISR() is a macro that calls xQueueGenericSendFromISR(), and therefore has the same return values as xQueueSendToFrontFromISR(). However, pdPASS is the only value that can be returned because xQueueOverwriteFromISR() will write to the queue even when the queue is already full.

### xQueueSendFromISR
(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)
This is a macro that calls xQueueGenericSendFromISR(). It is included for backward compatibility with versions of FreeRTOS.org that did not include the xQueueSendToBackFromISR() and xQueueSendToFromFrontFromISR() macros.

Post an item to the back of a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):
```c
void vBufferISR( void )
{
    char cIn;
    BaseType_t xHigherPriorityTaskWoken;

    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWoken = pdFALSE;

    // Loop until the buffer is empty.
    do
    {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

        // Post the byte.
        xQueueSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );
    } while( portINPUT_BYTE( BUFFER_COUNT ) );

    // Now the buffer is empty we can switch context if necessary.
    if( xHigherPriorityTaskWoken )
    {
        // Actual macro used here is port specific.
        portYIELD_FROM_ISR();
    }
}
```

### 参数
- **xQueue** - The handle to the queue on which the item is to be posted.
- **pvItemToQueue** - A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- **pxHigherPriorityTaskWoken** - [out] xQueueSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

### 返回
- pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

### xQueueReset(xQueue)
Reset a queue back to its original empty state. The return value is now obsolete and is always set to pdPASS.

### Type Definitions

- typedef struct QueueDefinition *QueueHandle_t
- typedef struct QueueDefinition *QueueSetHandle_t
  
  Type by which queue sets are referenced. For example, a call to xQueueCreateSet() returns an xQueueSet variable that can then be used as a parameter to xQueueSelectFromSet(), xQueueAddToSet(), etc.

- typedef struct QueueDefinition *QueueSetMemberHandle_t
  
  Queue sets can contain both queues and semaphores, so the QueueSetMemberHandle_t is defined as a type to be used where a parameter or return value can be either an QueueHandle_t or an SemaphoreHandle_t.

### Semaphore API

### Header File
Macros

- `semBINARY_SEMAPHORE_QUEUE_LENGTH`
- `semSEMAPHORE_QUEUE_ITEM_LENGTH`
- `semGIVE_BLOCK_TIME`

`vSemaphoreCreateBinary` (xSemaphore)

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a binary semaphore! [https://www.FreeRTOS.org/RTOS-task-notifications.html](https://www.FreeRTOS.org/RTOS-task-notifications.html)

This old `vSemaphoreCreateBinary`() macro is now deprecated in favour of the `xSemaphoreCreateBinary()` function. Note that binary semaphores created using the `vSemaphoreCreateBinary`() macro are created in a state such that the first call to ‘take’ the semaphore would pass, whereas binary semaphores created using `xSemaphoreCreateBinary()` are created in a state such that the semaphore must first be ‘given’ before it can be ‘taken’.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously ‘give’ the semaphore while another continuously ‘takes’ the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see `xSemaphoreCreateMutex()`.

Example usage:

```c
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to vSemaphoreCreateBinary ().
    // This is a macro so pass the variable in directly.
    vSemaphoreCreateBinary( xSemaphore );

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}
```

参数

- `xSemaphore` – Handle to the created semaphore. Should be of type `SemaphoreHandle_t`.

`xSemaphoreCreateBinary`()

Creates a new binary semaphore instance, and returns a handle by which the new semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a binary semaphore! [https://www.FreeRTOS.org/RTOS-task-notifications.html](https://www.FreeRTOS.org/RTOS-task-notifications.html)

Internally, within the FreeRTOS implementation, binary semaphores use a block of memory, in which the semaphore structure is stored. If a binary semaphore is created using `xSemaphoreCreateBinary()` then the
required memory is automatically dynamically allocated inside the xSemaphoreCreateBinary() function. (see https://www.FreeRTOS.org/a00111.html). If a binary semaphore is created using xSemaphoreCreateBinaryStatic() then the application writer must provide the memory. xSemaphoreCreateBinaryStatic() therefore allows a binary semaphore to be created without using any dynamic memory allocation.

The old vSemaphoreCreateBinary() macro is now deprecated in favour of this xSemaphoreCreateBinary() function. Note that binary semaphores created using the vSemaphoreCreateBinary() macro are created in a state such that the first call to ‘take’ the semaphore would pass, whereas binary semaphores created using xSemaphoreCreateBinary() are created in a state such that the the semaphore must first be ‘given’ before it can be ‘taken’.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously ‘give’ the semaphore while another continuously ‘takes’ the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see xSemaphoreCreateMutex().

Example usage:

```c
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to xSemaphoreCreateBinary().
    // This is a macro so pass the variable in directly.
    xSemaphore = xSemaphoreCreateBinary();

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}
```

xSemaphoreCreateBinaryStatic (pxStaticSemaphore)

Creates a new binary semaphore instance, and returns a handle by which the new semaphore can be referenced.

NOTE: In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a binary semaphore! https://www.FreeRTOS.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, binary semaphores use a block of memory, in which the semaphore structure is stored. If a binary semaphore is created using xSemaphoreCreateBinary() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateBinary() function. (see https://www.FreeRTOS.org/a00111.html). If a binary semaphore is created using xSemaphoreCreateBinaryStatic() then the application writer must provide the memory. xSemaphoreCreateBinaryStatic() therefore allows a binary semaphore to be created without using any dynamic memory allocation.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously ‘give’ the semaphore while another continuously ‘takes’ the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see xSemaphoreCreateMutex().

Example usage:
Chapter 2. API

SemaphoreHandle_t xSemaphore = NULL;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to xSemaphoreCreateBinaryStatic().
    // The semaphore's data structures will be placed in the xSemaphoreBuffer
    // variable, the address of which is passed into the function. The
    // function's parameter is not NULL, so the function will not attempt any
    // dynamic memory allocation, and therefore the function will not return
    // return NULL.
    xSemaphore = xSemaphoreCreateBinaryStatic( &xSemaphoreBuffer );

    // Rest of task code goes here.
}

参数

• pxStaticSemaphore – Must point to a variable of type StaticSemaphore_t, which will
  then be used to hold the semaphore’s data structure, removing the need for the memory
  to be allocated dynamically.

返回 If the semaphore is created then a handle to the created semaphore is returned. If
pxSemaphoreBuffer is NULL then NULL is returned.

xSemaphoreTake(xSemaphore, xBlockTime)

Macro to obtain a semaphore. The semaphore must have previously been created with a call to xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting().

Example usage:

SemaphoreHandle_t xSemaphore = NULL;

// A task that creates a semaphore.
void vATask( void * pvParameters )
{
    // Create the semaphore to guard a shared resource.
    xSemaphore = xSemaphoreCreateBinary();
}

// A task that uses the semaphore.
void vAnotherTask( void * pvParameters )
{
    // ... Do other things.

    if( xSemaphore != NULL )
    {
        // See if we can obtain the semaphore. If the semaphore is not available
        // wait 10 ticks to see if it becomes free.
        if( xSemaphoreTake( xSemaphore, ( TickType_t ) 10 ) == pdTRUE )
        {
            // We were able to obtain the semaphore and can now access the
            // shared resource.
            // ...
            // We have finished accessing the shared resource. Release the
            // semaphore.
            xSemaphoreGive( xSemaphore );
        }
        else
            // ..
    }
}
// We could not obtain the semaphore and can therefore not access
// the shared resource safely.
}
}

参数

- **xSemaphore** - A handle to the semaphore being taken - obtained when the semaphore was created.
- **xBlockTime** - The time in ticks to wait for the semaphore to become available. The macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. A block time of portMAX_DELAY can be used to block indefinitely (provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h).

返回 pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the semaphore becoming available.

**xSemaphoreTakeRecursive**(xMutex, xBlockTime)

*Macro* to recursively obtain, or ‘take’, a mutex type semaphore. The mutex must have previously been created using a call to xSemaphoreCreateRecursiveMutex();

configUSE_RECURSIVE_MUTEXES must be set to 1 in FreeRTOSConfig.h for this macro to be available.

This macro must not be used on mutexes created using xSemaphoreCreateMutex().

A mutex used recursively can be ‘taken’ repeatedly by the owner. The mutex doesn’t become available again until the owner has called xSemaphoreGiveRecursive() for each successful ‘take’ request. For example, if a task successfully ‘takes’ the same mutex 5 times then the mutex will not be available to any other task until it has also ‘given’ the mutex back exactly five times.

Example usage:

```c
SemaphoreHandle_t xMutex = NULL;

// A task that creates a mutex.
void vATask( void * pvParameters )
{
    // Create the mutex to guard a shared resource.
    xMutex = xSemaphoreCreateRecursiveMutex();
}

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
    // ... Do other things.
    if( xMutex != NULL )
    {
        // See if we can obtain the mutex. If the mutex is not available
        // wait 10 ticks to see if it becomes free.
        if( xSemaphoreTakeRecursive( xSemaphore, ( TickType_t ) 10 ) == pdTRUE )
        {
            // We were able to obtain the mutex and can now access the
            // shared resource.
            // ...
            // For some reason due to the nature of the code further calls to
            // xSemaphoreTakeRecursive() are made on the same mutex. In real
            // code these would not be just sequential calls as this would make
```
// no sense. Instead the calls are likely to be buried inside
// a more complex call structure.
xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

// The mutex has now been 'taken' three times, so will not be
// available to another task until it has also been given back
// three times. Again it is unlikely that real code would have
// these calls sequentially, but instead buried in a more complex
// call structure. This is just for illustrative purposes.
xSemaphoreGiveRecursive( xMutex );
xSemaphoreGiveRecursive( xMutex );
xSemaphoreGiveRecursive( xMutex );

// Now the mutex can be taken by other tasks.
}
else
{
    // We could not obtain the mutex and can therefore not access
    // the shared resource safely.
}

// A handle to the mutex being obtained. This is the handle returned by
// xSemaphoreCreateRecursiveMutex();
• xMutex

• xBlockTime - The time in ticks to wait for the semaphore to become available. The
  macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time
  of zero can be used to poll the semaphore. If the task already owns the semaphore then
  xSemaphoreTakeRecursive() will return immediately no matter what the value of xBlock-
  Time.

返回 pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the
semaphore becoming available.

xSemaphoreGive(xSemaphore)

Macro to release a semaphore. The semaphore must have previously been created with a call to
xSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting(), and obtained using
xSemaphoreTake().

This macro must not be used from an ISR. See xSemaphoreGiveFromISR() for an alternative which can be
used from an ISR.

This macro must also not be used on semaphores created using xSemaphoreCreateRecursiveMutex().

Example usage:

SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
    // Create the semaphore to guard a shared resource.
    xSemaphore = vSemaphoreCreateBinary();

    if( xSemaphore != NULL )
    {
        if( xSemaphoreGive( xSemaphore ) != pdTRUE )
        {
            // More code here
        }
    }
}
// We would expect this call to fail because we cannot give
// a semaphore without first "taking" it!
}

// Obtain the semaphore - don't block if the semaphore is not
// immediately available.
if( xSemaphoreTake( xSemaphore, ( TickType_t ) 0 ) )
{
    // We now have the semaphore and can access the shared resource.
    // ...

    // We have finished accessing the shared resource so can free the
    // semaphore.
    if( xSemaphoreGive( xSemaphore ) != pdTRUE )
    {
        // We would not expect this call to fail because we must have
        // obtained the semaphore to get here.
    }
}

参数

- xSemaphore - A handle to the semaphore being released. This is the handle returned
  when the semaphore was created.

返回 pdTRUE if the semaphore was released. pdFALSE if an error occurred. Semaphores are
implemented using queues. An error can occur if there is no space on the queue to post a
message - indicating that the semaphore was not first obtained correctly.

xSemaphoreGiveRecursive (xMutex)

Macro to recursively release, or ‘give’, a mutex type semaphore. The mutex must have previously been
created using a call to xSemaphoreCreateRecursiveMutex();

configUSE_RECURSIVE_MUTEXES must be set to 1 in FreeRTOSConfig.h for this macro to be available.

This macro must not be used on mutexes created using xSemaphoreCreateMutex().

A mutex used recursively can be ‘taken’ repeatedly by the owner. The mutex doesn’t become available
again until the owner has called xSemaphoreGiveRecursive() for each successful ‘take’ request. For example,
if a task successfully ‘takes’ the same mutex 5 times then the mutex will not be available to any other task
until it has also ‘given’ the mutex back exactly five times.

Example usage:

SemaphoreHandle_t xMutex = NULL;

// A task that creates a mutex.
void vATask( void * pvParameters )
{
    // Create the mutex to guard a shared resource.
    xMutex = xSemaphoreCreateRecursiveMutex();
}

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
    // ... Do other things.
if ( xMutex != NULL )
{
    // See if we can obtain the mutex. If the mutex is not available
    // wait 10 ticks to see if it becomes free.
    if ( xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 ) == pdTRUE )
    {
        // We were able to obtain the mutex and can now access the
        // shared resource.
        // ...
        // For some reason due to the nature of the code further calls to
        // xSemaphoreTakeRecursive() are made on the same mutex. In real
        // code these would not be just sequential calls as this would make
        // no sense. Instead the calls are likely to be buried inside
        // a more complex call structure.
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
        xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

        // The mutex has now been 'taken' three times, so will not be
        // available to another task until it has also been given back
        // three times. Again it is unlikely that real code would have
        // these calls sequentially, it would be more likely that the calls
        // to xSemaphoreGiveRecursive() would be called as a call stack
        // unwound. This is just for demonstrative purposes.
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );
        xSemaphoreGiveRecursive( xMutex );

        // Now the mutex can be taken by other tasks.
    }
    else
    {
        // We could not obtain the mutex and can therefore not access
        // the shared resource safely.
    }
}

参数
• xMutex - A handle to the mutex being released, or ‘given’. This is the handle returned
  by xSemaphoreCreateMutex().
返回 pdTRUE if the semaphore was given.

xSemaphoreGiveFromISR( xSemaphore, pxHigherPriorityTaskWoken )
Macro to release a semaphore. The semaphore must have previously been created with a call to
xSemaphoreCreateBinary() or xSemaphoreCreateCounting().

Mutex type semaphores (those created using a call to xSemaphoreCreateMutex()) must not be used with this
macro.
This macro can be used from an ISR.

Example usage:

#define LONG_TIME 0xffff
#define TICKS_TO_WAIT 10
SemaphoreHandle_t xSemaphore = NULL;
// Repetitive task.
void vATask( void *pvParameters )
{
    for( ;; )
    {
        // We want this task to run every 10 ticks of a timer. The semaphore
        // was created before this task was started.

        // Block waiting for the semaphore to become available.
        if( xSemaphoreTake( xSemaphore, LONG_TIME ) == pdTRUE )
        {
            // It is time to execute.
            // ...

            // We have finished our task. Return to the top of the loop where
            // we will block on the semaphore until it is time to execute
            // again. Note when using the semaphore for synchronisation with an
            // ISR in this manner there is no need to 'give' the semaphore back.
        }
    }
}

// Timer ISR
void vTimerISR( void *pvParameters )
{
    static uint8_t ucLocalTickCount = 0;
    static BaseType_t xHigherPriorityTaskWoken;

    // A timer tick has occurred.
    // ... Do other time functions.

    // Is it time for vATask () to run?
    xHigherPriorityTaskWoken = pdFALSE;
    ucLocalTickCount ++;
    if( ucLocalTickCount >= TICKS_TO_WAIT )
    {
        // Unblock the task by releasing the semaphore.
        xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );

        // Reset the count so we release the semaphore again in 10 ticks time.
        ucLocalTickCount = 0;
    }

    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // We can force a context switch here. Context switching from an
        // ISR uses port specific syntax. Check the demo task for your port
        // to find the syntax required.
    }
}

参数

- xSemaphore - A handle to the semaphore being released. This is the handle returned when the semaphore was created.
- pxHigherPriorityTaskWoken - xSemaphoreGiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if giving the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xSemaphoreGiveFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

返回 pdTRUE if the semaphore was successfully given, otherwise errQUEUE_FULL.
**xSemaphoreTakeFromISR (xSemaphore, pxHigherPriorityTaskWoken)**

*Macro* to take a semaphore from an ISR. The semaphore must have previously been created with a call to xSemaphoreCreateBinary() or xSemaphoreCreateCounting().

Mutex type semaphores (those created using a call to xSemaphoreCreateMutex()) must not be used with this macro.

This macro can be used from an ISR, however taking a semaphore from an ISR is not a common operation. It is likely to only be useful when taking a counting semaphore when an interrupt is obtaining an object from a resource pool (when the semaphore count indicates the number of resources available).

**参数**

- **xSemaphore** - A handle to the semaphore being taken. This is the handle returned when the semaphore was created.
- **pxHigherPriorityTaskWoken** -[out] xSemaphoreTakeFromISR() will set *px-HigherPriorityTaskWoken to pdTRUE if taking the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xSemaphoreTakeFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

**返回** pdTRUE if the semaphore was successfully taken, otherwise pdFALSE

**xSemaphoreCreateMutex ()**

Creates a new mutex type semaphore instance, and returns a handle by which the new mutex can be referenced.

Internally, within the FreeRTOS implementation, mutex semaphores use a block of memory, in which the mutex structure is stored. If a mutex is created using xSemaphoreCreateMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateMutex() function. (see https://www.FreeRTOS.org/a00111.html). If a mutex is created using xSemaphoreCreateMutexStatic() then the application writer must provided the memory. xSemaphoreCreateMutexStatic() therefore allows a mutex to be created without using any dynamic memory allocation.

Mutexes created using this function can be accessed using the xSemaphoreTake() and xSemaphoreGive() macros. The xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros must not be used.

This type of semaphore uses a priority inheritance mechanism so a task ‘taking’ a semaphore MUST ALWAYS ‘give’ the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always ‘gives’ the semaphore and another always ‘takes’ the semaphore) and from within interrupt service routines.

**Example usage:**

```c
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to xSemaphoreCreateMutex().
    // This is a macro so pass the variable in directly.
    xSemaphore = xSemaphoreCreateMutex();

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}
```

**返回** If the mutex was successfully created then a handle to the created semaphore is returned. If there was not enough heap to allocate the mutex data structures then NULL is returned.
xSemaphoreCreateMutexStatic(pxMutexBuffer)

Creates a new mutex type semaphore instance, and returns a handle by which the new mutex can be referenced.

Internally, within the FreeRTOS implementation, mutex semaphores use a block of memory, in which the mutex structure is stored. If a mutex is created using xSemaphoreCreateMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateMutex() function. (see https://www.FreeRTOS.org/a00111.html). If a mutex is created using xSemaphoreCreateMutexStatic() then the application writer must provide the memory. xSemaphoreCreateMutexStatic() therefore allows a mutex to be created without using any dynamic memory allocation.

Mutexes created using this function can be accessed using the xSemaphoreTake() and xSemaphoreGive() macros. The xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros must not be used.

This type of semaphore uses a priority inheritance mechanism so a task ‘taking’ a semaphore MUST ALWAYS ‘give’ the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always ‘gives’ the semaphore and another always ‘takes’ the semaphore) and from within interrupt service routines.

Example usage:

```c
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xMutexBuffer;

void vATask ( void * pvParameters )
{
    // A mutex cannot be used before it has been created. xMutexBuffer is
    // into xSemaphoreCreateMutexStatic() so no dynamic memory allocation is
    // attempted.
    xSemaphore = xSemaphoreCreateMutexStatic( &xMutexBuffer );

    // As no dynamic memory allocation was performed, xSemaphore cannot be NULL,
    // so there is no need to check it.
}
```

参数

- **pxMutexBuffer** - Must point to a variable of type StaticSemaphore_t, which will be used to hold the mutex’s data structure, removing the need for the memory to be allocated dynamically.

返回

- If the mutex was successfully created then a handle to the created mutex is returned. If pxMutexBuffer was NULL then NULL is returned.

xSemaphoreCreateCounting(uxMaxCount, uxInitialCount)

Creates a new recursive mutex type semaphore instance, and returns a handle by which the new recursive mutex can be referenced.

Internally, within the FreeRTOS implementation, recursive mutexes use a block of memory, in which the mutex structure is stored. If a recursive mutex is created using xSemaphoreCreateRecursiveMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateRecursiveMutex() function. (see https://www.FreeRTOS.org/a00111.html). If a recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic() then the application writer must provide the memory that will get used by the mutex. xSemaphoreCreateRecursiveMutexStatic() therefore allows a recursive mutex to be created without using any dynamic memory allocation.

Mutexes created using this macro can be accessed using the xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros. The xSemaphoreTake() and xSemaphoreGive() macros must not be used.

A mutex used recursively can be ‘taken’ repeatedly by the owner. The mutex doesn’t become available again until the owner has called xSemaphoreGiveRecursive() for each successful ‘take’ request. For example,
if a task successfully ‘takes’ the same mutex 5 times then the mutex will not be available to any other task until it has also ‘given’ the mutex back exactly five times.

This type of semaphore uses a priority inheritance mechanism so a task ‘taking’ a semaphore MUST ALWAYS ‘give’ the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See `xSemaphoreCreateBinary()` for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always ‘gives’ the semaphore and another always ‘takes’ the semaphore) and from within interrupt service routines.

Example usage:

```c
SemaphoreHandle_t xSemaphore;
void vATask( void * pvParameters )
{
  // Semaphore cannot be used before a call to xSemaphoreCreateMutex().
  // This is a macro so pass the variable in directly.
  xSemaphore = xSemaphoreCreateRecursiveMutex();
  if( xSemaphore != NULL )
  {
    // The semaphore was created successfully.
    // The semaphore can now be used.
  }
}
```

Creates a new recursive mutex type semaphore instance, and returns a handle by which the new recursive mutex can be referenced.

Internally, within the FreeRTOS implementation, recursive mutexes use a block of memory, in which the mutex structure is stored. If a recursive mutex is created using `xSemaphoreCreateRecursiveMutex()` then the required memory is automatically dynamically allocated inside the `xSemaphoreCreateRecursiveMutex()` function. (see [https://www.FreeRTOS.org/a00111.html](https://www.FreeRTOS.org/a00111.html)). If a recursive mutex is created using `xSemaphoreCreateRecursiveMutexStatic()` then the application writer must provide the memory that will get used by the mutex. `xSemaphoreCreateRecursiveMutexStatic()` therefore allows a recursive mutex to be created without using any dynamic memory allocation.

Mutexes created using this macro can be accessed using the `xSemaphoreTakeRecursive()` and `xSemaphoreGiveRecursive()` macros. The `xSemaphoreTake()` and `xSemaphoreGive()` macros must not be used.

A mutex used recursively can be ‘taken’ repeatedly by the owner. The mutex doesn’t become available again until the owner has called `xSemaphoreGiveRecursive()` for each successful ‘take’ request. For example, if a task successfully ‘takes’ the same mutex 5 times then the mutex will not be available to any other task until it has also ‘given’ the mutex back exactly five times.

This type of semaphore uses a priority inheritance mechanism so a task ‘taking’ a semaphore MUST ALWAYS ‘give’ the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See `xSemaphoreCreateBinary()` for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always ‘gives’ the semaphore and another always ‘takes’ the semaphore) and from within interrupt service routines.

Example usage:

```c
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xMutexBuffer;
```
void vATask( void * pvParameters )
{
    // A recursive semaphore cannot be used before it is created. Here a
    // recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic().
    // The address of xMutexBuffer is passed into the function, and will hold
    // the mutexes data structures - so no dynamic memory allocation will be
    // attempted.
    xSemaphore = xSemaphoreCreateRecursiveMutexStatic( &xMutexBuffer );
    // As no dynamic memory allocation was performed, xSemaphore cannot be NULL,
    // so there is no need to check it.
}

Creates a new counting semaphore instance, and returns a handle by which the new counting semaphore can
be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a

Internally, within the FreeRTOS implementation, counting semaphores use a block of memory, in which
the counting semaphore structure is stored. If a counting semaphore is created using xSemaphoreCreate-
Counting() then the required memory is automatically dynamically allocated inside the xSemaphoreCreate-
Counting() function. (see [https://www.FreeRTOS.org/a00111.html](https://www.FreeRTOS.org/a00111.html)). If a counting semaphore is created using xSemaphoreCreateCountingStatic() then the application writer can instead optionally provide the memory
that will get used by the counting semaphore. xSemaphoreCreateCountingStatic() therefore allows a counting
semaphore to be created without using any dynamic memory allocation.

Counting semaphores are typically used for two things:

1) Counting events.

In this usage scenario an event handler will ‘give’ a semaphore each time an event occurs (incrementing
the semaphore count value), and a handler task will ‘take’ a semaphore each time it processes an event
(decrementing the semaphore count value). The count value is therefore the difference between the number of
events that have occurred and the number that have been processed. In this case it is desirable for the initial
count value to be zero.

2) Resource management.

In this usage scenario the count value indicates the number of resources available. To obtain control of a
resource a task must first obtain a semaphore - decrementing the semaphore count value. When the count value
reaches zero there are no free resources. When a task finishes with the resource it ‘gives’ the semaphore back
- incrementing the semaphore count value. In this case it is desirable for the initial count value to be equal to
the maximum count value, indicating that all resources are free.

Example usage:

SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
    SemaphoreHandle_t xSemaphore = NULL;
    // Semaphore cannot be used before a call to xSemaphoreCreateCounting().
    // The max value to which the semaphore can count should be 10, and the
    // initial value assigned to the count should be 0.
    xSemaphore = xSemaphoreCreateCounting( 10, 0 );
    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
    }
}
// The semaphore can now be used.

return xSemaphore Handle to the created mutex semaphore. Should be of type SemaphoreHandle_t.

参数

- pxStaticSemaphore - Must point to a variable of type StaticSemaphore_t, which will then be used to hold the recursive mutex’s data structure, removing the need for the memory to be allocated dynamically.
- uxMaxCount - The maximum count value that can be reached. When the semaphore reaches this value it can no longer be ‘given’.
- uxInitialCount - The count value assigned to the semaphore when it is created.

返回

- If the recursive mutex was successfully created then a handle to the created recursive mutex is returned. If pxMutexBuffer was NULL then NULL is returned.
- Handle to the created semaphore. Null if the semaphore could not be created.

xSemaphoreCreateCountingStatic(uxMaxCount, uxInitialCount, pxSemaphoreBuffer)

Creates a new counting semaphore instance, and returns a handle by which the new counting semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a counting semaphore! [https://www.FreeRTOS.org/RTOS-task-notifications.html](https://www.FreeRTOS.org/RTOS-task-notifications.html)

Internally, within the FreeRTOS implementation, counting semaphores use a block of memory, in which the counting semaphore structure is stored. If a counting semaphore is created using xSemaphoreCreateCounting() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateCounting() function. (see [https://www.FreeRTOS.org/a00111.html](https://www.FreeRTOS.org/a00111.html)). If a counting semaphore is created using xSemaphoreCreateCountingStatic() then the application writer must provide the memory. xSemaphoreCreateCountingStatic() therefore allows a counting semaphore to be created without using any dynamic memory allocation.

Counting semaphores are typically used for two things:

1) Counting events.

In this usage scenario an event handler will ‘give’ a semaphore each time an event occurs (incrementing the semaphore count value), and a handler task will ‘take’ a semaphore each time it processes an event (decrementing the semaphore count value). The count value is therefore the difference between the number of events that have occurred and the number that have been processed. In this case it is desirable for the initial count value to be zero.

2) Resource management.

In this usage scenario the count value indicates the number of resources available. To obtain control of a resource a task must first obtain a semaphore - decrementing the semaphore count value. When the count value reaches zero there are no free resources. When a task finishes with the resource it ‘gives’ the semaphore back - incrementing the semaphore count value. In this case it is desirable for the initial count value to be equal to the maximum count value, indicating that all resources are free.

Example usage:

```c
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters )
{
SemaphoreHandle_t xSemaphore = NULL;
```
// Counting semaphore cannot be used before they have been created. Create
// a counting semaphore using xSemaphoreCreateCountingStatic(). The max
// value to which the semaphore can count is 10, and the initial value
// assigned to the count will be 0. The address of xSemaphoreBuffer is
// passed in and will be used to hold the semaphore structure, so no dynamic
// memory allocation will be used.
xSemaphore = xSemaphoreCreateCounting( 10, 0, &xSemaphoreBuffer );

// No memory allocation was attempted so xSemaphore cannot be NULL, so there
// is no need to check its value.

参数

• **uxMaxCount** - The maximum count value that can be reached. When the semaphore
  reaches this value it can no longer be ‘given’.
• **uxInitialCount** - The count value assigned to the semaphore when it is created.
• **pxSemaphoreBuffer** - Must point to a variable of type StaticSemaphore_t, which will
  then be used to hold the semaphore’s data structure, removing the need for the memory
  to be allocated dynamically.

返回

If the counting semaphore was successfully created then a handle to the created counting
semaphore is returned. If pxSemaphoreBuffer was NULL then NULL is returned.

vSemaphoreDelete (xSemaphore)

Delete a semaphore. This function must be used with care. For example, do not delete a mutex type semaphore
if the mutex is held by a task.

参数

• **xSemaphore** - A handle to the semaphore to be deleted.

xSemaphoreGetMutexHolder (xSemaphore)

If xMutex is indeed a mutex type semaphore, return the current mutex holder. If xMutex is not a mutex type
semaphore, or the mutex is available (not held by a task), return NULL.

Note: This is a good way of determining if the calling task is the mutex holder, but not a good way of deter-
mining the identity of the mutex holder as the holder may change between the function exiting and the returned
value being tested.

xSemaphoreGetMutexHolderFromISR (xSemaphore)

If xMutex is indeed a mutex type semaphore, return the current mutex holder. If xMutex is not a mutex type
semaphore, or the mutex is available (not held by a task), return NULL.

uxSemaphoreGetCount (xSemaphore)

If the semaphore is a counting semaphore then uxSemaphoreGetCount() returns its current count value. If the
semaphore is a binary semaphore then uxSemaphoreGetCount() returns 1 if the semaphore is available, and 0
if the semaphore is not available.

Type Definitions

typedef QueueHandle_t SemaphoreHandle_t

Timer API

Header File

• components/freertos/FreeRTOS-Kernel/include/freertos/timers.h
Chapter 2. API

## Functions

**TimerHandle_t xTimerCreate** (const char *const pcTimerName, const TickType_t xTimerPeriodInTicks, const UBaseType_t uxAutoReload, void *const pvTimerID, TimerCallbackFunction_t pxCallbackFunction)

`TimerHandle_t xTimerCreate( const char * const pcTimerName, TickType_t xTimerPeriodInTicks, UBaseType_t uxAutoReload, void * pvTimerID, TimerCallbackFunction_t pxCallbackFunction);`

Creates a new software timer instance, and returns a handle by which the created software timer can be referenced.

Internally, within the FreeRTOS implementation, software timers use a block of memory, in which the timer data structure is stored. If a software timer is created using xTimerCreate() then the required memory is automatically dynamically allocated inside the xTimerCreate() function. (see [https://www.FreeRTOS.org/a00111.](https://www.FreeRTOS.org/a00111.html)). If a software timer is created using xTimerCreateStatic() then the application writer must provide the memory that will get used by the software timer. xTimerCreateStatic() therefore allows a software timer to be created without using any dynamic memory allocation.

Timers are created in the dormant state. The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() and xTimerChangePeriodFromISR() API functions can all be used to transition a timer into the active state.

### Example usage:

```c
* #define NUM_TIMERS 5
* 
* // An array to hold handles to the created timers.
* TimerHandle_t xTimers[ NUM_TIMERS ];
* 
* // An array to hold a count of the number of times each timer expires.
* int32_t lExpireCounters[ NUM_TIMERS ] = { 0 };
* 
* // Define a callback function that will be used by multiple timer instances.
* // The callback function does nothing but count the number of times the
* // associated timer expires, and stop the timer once the timer has expired
* // 10 times.
* void vTimerCallback( TimerHandle_t pxTimer )
* {
*   int32_t lArrayIndex;
*   const int32_t xMaxExpiryCountBeforeStopping = 10;
*   
*   // Optionally do something if the pxTimer parameter is NULL.
*   configASSERT( pxTimer );
*   
*   // Which timer expired?
*   lArrayIndex = ( int32_t ) pvTimerGetTimerID( pxTimer );
*   
*   // Increment the number of times that pxTimer has expired.
*   lExpireCounters[ lArrayIndex ] += 1;
*   
*   // If the timer has expired 10 times then stop it from running.
*   if( lExpireCounters[ lArrayIndex ] == xMaxExpiryCountBeforeStopping )
*     {
*       // Do not use a block time if calling a timer API function from a
*       // timer callback function, as doing so could cause a deadlock!
*       xTimerStop( pxTimer, 0 );
*     }
* } 
* 
* void main( void )
* {
*   int32_t x;
```
// Create then start some timers. Starting the timers before the scheduler
// has been started means the timers will start running immediately that
// the scheduler starts.
for( x = 0; x < NUM_TIMERS; x++ )
{
    xTimers[ x ] = xTimerCreate( "Timer", // Just a text name,
        ( 100 * x ), // The timer period
        pdTRUE, // The timers will auto-reload themselves when they expire.
        ( void * ) x, // Assign each timer a unique id equal to its array index.
        vTimerCallback // Each timer calls the same callback when it expires.
    );

    if( xTimers[ x ] == NULL )
        // The timer was not created.
    }
else
    // Start the timer. No block time is specified, and even if one was
    // it would be ignored because the scheduler has not yet been started.
    if( xTimerStart( xTimers[ x ], 0 ) != pdPASS )
        // The timer could not be set into the Active state.
}

// ... Create tasks here.
// ... Starting the scheduler will start the timers running as they have already
// been set into the active state.
vTaskStartScheduler();

// Should not reach here.
for( ;; );
}

参数

- pcTimerName - A text name that is assigned to the timer. This is done purely to assist debugging. The kernel itself only ever references a timer by its handle, and never by its name.

- xTimerPeriodInTicks - The timer period. The time is defined in tick periods so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xTimerPeriodInTicks should be set to 100. Alternatively, if the timer must expire after 500ms, then xPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000. Time timer period must be greater than 0.
• **uxAutoReload** – If `uxAutoReload` is set to `pTRUE` then the timer will expire repeatedly with a frequency set by the `xTimerPeriodInTicks` parameter. If `uxAutoReload` is set to `pFALSE` then the timer will be a one-shot timer and enter the dormant state after it expires.

• **pvTimerID** – An identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer.

• **pxCallbackFunction** – The function to call when the timer expires. Callback functions must have the prototype defined by `TimerCallbackFunction_t`, which is “void vCallbackFunction(TimerHandle_t xTimer);”.

If the timer is successfully created then a handle to the newly created timer is returned. If the timer cannot be created because there is insufficient FreeRTOS heap remaining to allocate the timer structures then `NULL` is returned.

### Example usage:

```c
static void prvTimerCallback( TimerHandle_t xExpiredTimer ) {
    UBaseType_t *puxVariableToIncrement;
    BaseType_t xReturned;

    // Obtain the address of the variable to increment from the timer ID.
    puxVariableToIncrement = ( UBaseType_t * )pvTimerGetTimerID( xExpiredTimer );

    // Increment the variable to show the timer callback has executed.
    ( puxVariableToIncrement )++;
}
```

Internally, within the FreeRTOS implementation, software timers use a block of memory, in which the timer data structure is stored. If a software timer is created using `xTimerCreate()` then the required memory is automatically dynamically allocated inside the `xTimerCreate()` function. (see [https://www.FreeRTOS.org/a00111.html](https://www.FreeRTOS.org/a00111.html)). If a software timer is created using `xTimerCreateStatic()` then the application writer must provide the memory that will get used by the software timer. `xTimerCreateStatic()` therefore allows a software timer to be created without using any dynamic memory allocation.

Timers are created in the dormant state. The `xTimerStart()`, `xTimerReset()`, `xTimerStartFromISR()`, `xTimerResetFromISR()`, `xTimerChangePeriod()` and `xTimerChangePeriodFromISR()` API functions can all be used to transition a timer into the active state.
void main( void )
{
  // Create the software time. xTimerCreateStatic() has an extra parameter
  // than the normal xTimerCreate() API function. The parameter is a
  // pointer
  // to the StaticTimer_t structure that will hold the software timer
  // structure. If the parameter is passed as NULL then the structure
  // will be
  // allocated dynamically, just as if xTimerCreate() had been called.
  xTimer = xTimerCreateStatic( "T1", 0, &uxVariableToIncrement, prvTimerCallback, &xTimerBuffer );
  // The buffer that will
  // hold the software timer structure.
  // The scheduler has not started yet so a block time is not used.
  xReturned = xTimerStart( xTimer, 0 );
  // ...
  // Create tasks here.
  // ...

  // Starting the scheduler will start the timers running as they have
  // already
  // been set into the active state.
  vTaskStartScheduler();

  // Should not reach here.
  for( ;; );
}

参数

- **pcTimerName** – A text name that is assigned to the timer. This is done purely to assist debugging. The kernel itself only ever references a timer by its handle, and never by its name.

- **xTimerPeriodInTicks** – The timer period. The time is defined in tick periods so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xTimerPeriodInTicks should be set to 100. Alternatively, if the timer must expire after 500ms, then xPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000. The timer period must be greater than 0.
• **uxAutoReload** – If uxAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriodInTicks parameter. If uxAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.

• **pvTimerID** – An identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer.

• **pxCallbackFunction** – The function to call when the timer expires. Callback functions must have the prototype defined by TimerCallbackFunction_t, which is “void vCallbackFunction(TimerHandle_t xTimer);”.

• **pxTimerBuffer** – Must point to a variable of type StaticTimer_t, which will be then be used to hold the software timer’s data structures, removing the need for the memory to be allocated dynamically.

If the timer is created then a handle to the created timer is returned. If pxTimerBuffer was NULL then NULL is returned.

```c
void *pvTimerGetTimerID (const TimerHandle_t xTimer)
void *pvTimerGetTimerID( TimerHandle_t xTimer );
```

Returns the ID assigned to the timer.

IDs are assigned to timers using the pvTimerID parameter of the call to xTimerCreated() that was used to create the timer, and by calling the vTimerSetTimerID() API function.

If the same callback function is assigned to multiple timers then the timer ID can be used as time specific (timer local) storage.

Example usage:

See the xTimerCreate() API function example usage scenario.

```c
参数 xTimer – The timer being queried.
返回 The ID assigned to the timer being queried.
```

```c
void vTimerSetTimerID (TimerHandle_t xTimer, void *pvNewID)
void vTimerSetTimerID( TimerHandle_t xTimer, void *pvNewID );
```

Sets the ID assigned to the timer.

IDs are assigned to timers using the pvTimerID parameter of the call to xTimerCreated() that was used to create the timer.

If the same callback function is assigned to multiple timers then the timer ID can be used as time specific (timer local) storage.

Example usage:

See the xTimerCreate() API function example usage scenario.

```c
参数
    • xTimer – The timer being updated.
    • pvNewID – The ID to assign to the timer.
```

```c
BaseType_t xTimerIsTimerActive (TimerHandle_t xTimer)
BaseType_t xTimerIsTimerActive( TimerHandle_t xTimer );
```

Queries a timer to see if it is active or dormant.

A timer will be dormant if: 1) It has been created but not started, or 2) It is an expired one-shot timer that has not been restarted.
Timers are created in the dormant state. The `xTimerStart()`, `xTimerReset()`, `xTimerStartFromISR()`, `xTimerResetFromISR()`, `xTimerChangePeriod()` and `xTimerChangePeriodFromISR()` API functions can all be used to transition a timer into the active state.

Example usage:

```c
// This function assumes xTimer has already been created.
void vAFunction( TimerHandle_t xTimer )
{
    if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and equivalently "if( xTimerIsTimerActive( xTimer ) )"
    {
        // xTimer is active, do something.
    }
    else
    {
        // xTimer is not active, do something else.
    }
}
```

**参数** xTimer — The timer being queried.

**返回** pdFALSE will be returned if the timer is dormant. A value other than pdFALSE will be returned if the timer is active.

**TaskHandle_t xTimerGetTimerDaemonTaskHandle** (void)

TaskHandle_t xTimerGetTimerDaemonTaskHandle( void );

Simply returns the handle of the timer service/daemon task. It it not valid to call `xTimerGetTimerDaemonTaskHandle()` before the scheduler has been started.

**BaseType_t xTimerPendFunctionCallFromISR** (PendedFunction_t xFunctionToPend, void *pvParameter1, uint32_t ulParameter2, BaseType_t *pxHigherPriorityTaskWoken);

BaseType_t xTimerPendFunctionCallFromISR( PendedFunction_t xFunctionToPend, void *pvParameter1, uint32_t ulParameter2, BaseType_t *pxHigherPriorityTaskWoken );

Used from application interrupt service routines to defer the execution of a function to the RTOS daemon task (the timer service task, hence this function is implemented in timers.c and is prefixed with ‘Timer’).

Ideally an interrupt service routine (ISR) is kept as short as possible, but sometimes an ISR either has a lot of processing to do, or needs to perform processing that is not deterministic. In these cases `xTimerPendFunctionCallFromISR()` can be used to defer processing of a function to the RTOS daemon task.

A mechanism is provided that allows the interrupt to return directly to the task that will subsequently execute the pended callback function. This allows the callback function to execute contiguously in time with the interrupt - just as if the callback had executed in the interrupt itself.

Example usage:

```c
/* // The callback function that will execute in the context of the daemon... task. */
/* // Note callback functions must all use this same prototype. */
void vProcessInterface( void *pvParameter1, uint32_t ulParameter2 )
{
    BaseType_t xInterfaceToService;
    /*
    // The interface that requires servicing is passed in the second
    */
```

(下页继续)
API

Chapter 2. API

* // parameter. The first parameter is not used in this case.
* xInterfaceToService = ( BaseType_t ) ulParameter2;
* }
* // ...Perform the processing here...
*
* // An ISR that receives data packets from multiple interfaces
* void vAnISR( void )
* {
*   BaseType_t xInterfaceToService, xHigherPriorityTaskWoken;
*   // Query the hardware to determine which interface needs processing.
*   xInterfaceToService = prvCheckInterfaces();
*   // The actual processing is to be deferred to a task. Request the
*   // vProcessInterface() callback function is executed, passing in the
*   // number of the interface that needs processing. The interface to
*   // service is passed in the second parameter. The first parameter is
*   // not used in this case.
*   xHigherPriorityTaskWoken = pdFALSE;
*   xTimerPendFunctionCallFromISR( vProcessInterface, NULL, ( uint32_t )
*                                xInterfaceToService, &xHigherPriorityTaskWoken );
*   // If xHigherPriorityTaskWoken is now set to pdTRUE then a context
*   // switch should be requested. The macro used is port specific and will
*   // be either portYIELD_FROM_ISR() or portEND_SWITCHING_ISR() - refer to
*   // the documentation page for the port being used.
*   portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
* }
*
**

参数

- **xFunctionToPend** - The function to execute from the timer service/daemon task. The function must conform to the PendedFunction_t prototype.
- **pvParameter1** - The value of the callback function’s first parameter. The parameter has a void * type to allow it to be used to pass any type. For example, unsigned longs can be cast to a void *, or the void * can be used to point to a structure.
- **ulParameter2** - The value of the callback function’s second parameter.
- **pxHigherPriorityTaskWoken** - As mentioned above, calling this function will result in a message being sent to the timer daemon task. If the priority of the timer daemon task (which is set using configTIMER_TASK_PRIORITY in FreeRTOSConfig.h) is higher than the priority of the currently running task (the task the interrupt interrupted) then *pxHigherPriorityTaskWoken will be set to pdTRUE within xTimerPendFunctionCallFromISR(), indicating that a context switch should be requested before the interrupt exits. For that reason *pxHigherPriorityTaskWoken must be initialised to pdFALSE. See the example code below.

返回 pdPASS is returned if the message was successfully sent to the timer daemon task, otherwise pdFALSE is returned.

```
BaseType_t xTimerPendFunctionCall ( PendedFunction_t xFunctionToPend, void *pvParameter1, uint32_t ulParameter2, TickType_t xTicksToWait)
```

Used to defer the execution of a function to the RTOS daemon task (the timer service task, hence this function is implemented in timers.c and is prefixed with ‘Timer’).
• **xFunctionToPend** – The function to execute from the timer service/daemon task. The function must conform to the PendedFunction_t prototype.

• **pvParameter1** – The value of the callback function’s first parameter. The parameter has a void * type to allow it to be used to pass any type. For example, unsigned longs can be cast to a void *, or the void * can be used to point to a structure.

• **ulParameter2** – The value of the callback function’s second parameter.

• **xTicksToWait** – Calling this function will result in a message being sent to the timer daemon task on a queue. xTicksToWait is the amount of time the calling task should remain in the Blocked state (so not using any processing time) for space to become available on the timer queue if the queue is found to be full.

返回 pdPASS is returned if the message was successfully sent to the timer daemon task, otherwise pdFALSE is returned.

```c
const char *pcTimerGetName(TimerHandle_t xTimer)
const char * const pcTimerGetName(TimerHandle_t xTimer);
```

Returns the name that was assigned to a timer when the timer was created.

参数 **xTimer** – The handle of the timer being queried.

返回 The name assigned to the timer specified by the xTimer parameter.

```c
void vTimerSetReloadMode(TimerHandle_t xTimer, const UBaseType_t uxAutoReload);
void vTimerSetReloadMode(TimerHandle_t xTimer, const UBaseType_t uxAutoReload);
```

Updates a timer to be either an auto-reload timer, in which case the timer automatically resets itself each time it expires, or a one-shot timer, in which case the timer will only expire once unless it is manually restarted.

参数

• **xTimer** – The handle of the timer being updated.

• **uxAutoReload** – If uxAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the timer’s period (see the xTimerPeriodInTicks parameter of the xTimerCreate() API function). If uxAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.

```c
UBaseType_t uxTimerGetReloadMode(TimerHandle_t xTimer)
UBaseType_t uxTimerGetReloadMode(TimerHandle_t xTimer);
```

Queries a timer to determine if it is an auto-reload timer, in which case the timer automatically resets itself each time it expires, or a one-shot timer, in which case the timer will only expire once unless it is manually restarted.

参数 **xTimer** – The handle of the timer being queried.

返回 If the timer is an auto-reload timer then pdTRUE is returned, otherwise pdFALSE is returned.

```c
TickType_t xTimerGetPeriod(TimerHandle_t xTimer)
```

Returns the period of a timer.

参数 **xTimer** – The handle of the timer being queried.

返回 The period of the timer in ticks.

```c
TickType_t xTimerGetExpiryTime(TimerHandle_t xTimer)
```

Returns the time in ticks at which the timer will expire. If this is less than the current tick count then the expiry time has overflowed from the current time.

参数 **xTimer** – The handle of the timer being queried.

返回 If the timer is running then the time in ticks at which the timer will next expire is returned.

If the timer is not running then the return value is undefined.
void vApplicationGetTimerTaskMemory(StaticTask_t **ppxTimerTaskTCBBuffer, StackType_t **ppxTimerTaskStackBuffer, uint32_t *pulTimerTaskStackSize)

This function is used to provide a statically allocated block of memory to FreeRTOS to hold the Timer Task TCB. This function is required when configSUPPORT_STATIC_ALLOCATION is set. For more information see this URI: https://www.FreeRTOS.org/a00110.html#configSUPPORT_STATIC_ALLOCATION

参数
- ppxTimerTaskTCBBuffer - A handle to a statically allocated TCB buffer
- ppxTimerTaskStackBuffer - A handle to a statically allocated Stack buffer for the idle task
- pulTimerTaskStackSize - A pointer to the number of elements that will fit in the allocated stack buffer

Macros

tmrCOMMAND_EXECUTE_CALLBACK_FROM_ISR

tmrCOMMAND_EXECUTE_CALLBACK

tmrCOMMAND_START_DONT_TRACE

tmrCOMMAND_START

tmrCOMMAND_RESET

tmrCOMMAND_STOP

tmrCOMMAND_CHANGE_PERIOD

tmrCOMMAND_DELETE

tmrFIRST_FROM_ISR_COMMAND

tmrCOMMAND_START_FROM_ISR

tmrCOMMAND_RESET_FROM_ISR

tmrCOMMAND_STOP_FROM_ISR

tmrCOMMAND_CHANGE_PERIOD_FROM_ISR

xTimerStart (xTimer, xTicksToWait)

BaseType_t xTimerStart( TimerHandle_t xTimer, TickType_t xTicksToWait );

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerStart() starts a timer that was previously created using the xTimerCreate() API function. If the timer had already been started and was already in the active state, then xTimerStart() has equivalent functionality to the xTimerReset() API function.
Starting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the mean time, the callback function associated with the timer will get called ‘n’ ticks after xTimerStart() was called, where ‘n’ is the timers defined period.

It is valid to call xTimerStart() before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when xTimerStart() was called.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerStart() to be available.

Example usage:
See the xTimerCreate() API function example usage scenario.

参数
• xTimer - The handle of the timer being started/restarted.
• xTicksToWait - Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the start command to be successfully sent to the timer command queue, should the queue already be full when xTimerStart() was called. xTicksToWait is ignored if xTimerStart() is called before the scheduler is started.

返回 pdFAIL will be returned if the start command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStart() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

xTimerStop (xTimer, xTicksToWait)
BaseType_t xTimerStop( TimerHandle_t xTimer, TickType_t xTicksToWait );

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerStop() stops a timer that was previously started using either of the The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() or xTimerChangePeriodFromISR() API functions.

Stopping a timer ensures the timer is not in the active state.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerStop() to be available.

Example usage:
See the xTimerCreate() API function example usage scenario.

参数
• xTimer - The handle of the timer being stopped.
• xTicksToWait - Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the stop command to be successfully sent to the timer command queue, should the queue already be full when xTimerStop() was called. xTicksToWait is ignored if xTimerStop() is called before the scheduler is started.

返回 pdFAIL will be returned if the stop command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.
**xTimerChangePeriod** (xTimer, xNewPeriod, xTicksToWait)

* BaseType_t xTimerChangePeriod( TimerHandle_t xTimer, TickType_t xNewPeriod, TickType_t xTicksToWait );

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerChangePeriod() changes the period of a timer that was previously created using the xTimerCreate() API function.

xTimerChangePeriod() can be called to change the period of an active or dormant state timer.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerChangePeriod() to be available.

Example usage:

```c
void vAFunction( TimerHandle_t xTimer ) {
    if( xTimerIsTimerActive( xTimer ) ) { // xTimer is already active - delete it.
        xTimerDelete( xTimer );
    } else { // xTimer is not active, change its period to 500ms. This will also cause the timer to start. Block for a maximum of 100 ticks if the change period command cannot immediately be sent to the timer command queue.
        if( xTimerChangePeriod( xTimer, 500 / portTICK_PERIOD_MS, 100 ) == pdPASS ) { // The command was successfully sent.
            } else { // The command could not be sent, even after waiting for 100 ticks to pass. Take appropriate action here.
            }
        }
    }
}
```

**Parameter**

- **xTimer** - The handle of the timer that is having its period changed.
- **xNewPeriod** - The new period for xTimer. Timer periods are specified in tick periods, so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xNewPeriod should be set to 100. Alternatively, if the timer must expire after 500ms, then xNewPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
• **xTicksToWait** – Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the change period command to be successfully sent to the timer command queue, should the queue already be full when `xTimerChangePeriod()` was called. `xTicksToWait` is ignored if `xTimerChangePeriod()` is called before the scheduler is started.

返回 pdFAIL will be returned if the change period command could not be sent to the timer command queue even after `xTicksToWait` ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the `configTIMER_TASK_PRIORITY` configuration constant.

**xTimerDelete** (xTimer, xTicksToWait)

BaseType_t `xTimerDelete(TimerHandle_t xTimer, TickType_t xTicksToWait);`

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the `configTIMER_QUEUE_LENGTH` configuration constant.

`xTimerDelete()` deletes a timer that was previously created using the `xTimerCreate()` API function.

The `configUSE_TIMERS` configuration constant must be set to 1 for `xTimerDelete()` to be available.

Example usage:

See the `xTimerChangePeriod()` API function example usage scenario.

参数

• **xTimer** – The handle of the timer being deleted.
• **xTicksToWait** – Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the delete command to be successfully sent to the timer command queue, should the queue already be full when `xTimerDelete()` was called. `xTicksToWait` is ignored if `xTimerDelete()` is called before the scheduler is started.

返回 pdFAIL will be returned if the delete command could not be sent to the timer command queue even after `xTicksToWait` ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the `configTIMER_TASK_PRIORITY` configuration constant.

**xTimerReset** (xTimer, xTicksToWait)

BaseType_t `xTimerReset(TimerHandle_t xTimer, TickType_t xTicksToWait);`

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the `configTIMER_QUEUE_LENGTH` configuration constant.

`xTimerReset()` re-starts a timer that was previously created using the `xTimerCreate()` API function. If the timer had already been started and was already in the active state, then `xTimerReset()` will cause the timer to re-evaluate its expiry time so that it is relative to when `xTimerReset()` was called. If the timer was in the dormant state then `xTimerReset()` has equivalent functionality to the `xTimerStart()` API function.

Resetting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the mean time, the callback function associated with the timer will get called ‘n’ ticks after `xTimerReset()` was called, where ‘n’ is the timers defined period.

It is valid to call `xTimerReset()` before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when `xTimerReset()` was called.
The configUSE_TIMERS configuration constant must be set to 1 for xTimerReset() to be available.

Example usage:

```c
// When a key is pressed, an LCD back-light is switched on. If 5 seconds pass
// without a key being pressed, then the LCD back-light is switched off. In this case, the timer is a one-shot timer.

TimerHandle_t xBacklightTimer = NULL;

// The callback function assigned to the one-shot timer. In this case the parameter is not used.
void vBacklightTimerCallback( TimerHandle_t pxTimer ) {
    // The timer expired, therefore 5 seconds must have passed since a key
    // was pressed. Switch off the LCD back-light.
    vSetBacklightState( BACKLIGHT_OFF );
}

// The key press event handler.
void vKeyPressEventHandler( char cKey ) {
    // Ensure the LCD back-light is on, then reset the timer that is responsible for turning the back-light off after 5 seconds of key inactivity. Wait 10 ticks for the command to be successfully sent if it cannot be sent immediately.
    vSetBacklightState( BACKLIGHT_ON );
    if( xTimerReset( xBacklightTimer, 100 ) != pdPASS ) {
        // The reset command was not executed successfully. Take appropriate action here.
    }
    // Perform the rest of the key processing here.
}

void main( void ) {
    int32_t x;
    // Create then start the one-shot timer that is responsible for turning the back-light off if no keys are pressed within a 5 second period.
    xBacklightTimer = xTimerCreate( "BacklightTimer", // Just a text name, not used by the kernel.
        ( 5000 / portTICK_PERIOD_MS), // The timer period in ticks.
        pdFALSE, // The timer is a one-shot timer.
        0, // The id is not used by the callback so can take any value.
        vBacklightTimerCallback // The callback function that switches the LCD back-light off.
    );
    if( xBacklightTimer == NULL ) {
        // The timer was not created.
    } else {
```
// Start the timer. No block time is specified, and even if one was
// it would be ignored because the scheduler has not yet been
// started.
if( xTimerStart( xBacklightTimer, 0 ) != pdPASS )
{
    // The timer could not be set into the Active state.
}

// ... Create tasks here. // ...

// Starting the scheduler will start the timer running as it has already
// been set into the active state.
vTaskStartScheduler();

// Should not reach here.
for( ;; );

参数
- **xTimer** - The handle of the timer being reset/started/restarted.
- **xTicksToWait** - Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the reset command to be successfully sent to the timer command queue, should the queue already be full when xTimerReset() was called. xTicksToWait is ignored if xTimerReset() is called before the scheduler is started.

返回 pdFAIL will be returned if the reset command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/demon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStart() is actually called. The timer service/demon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

**xTimerStartFromISR** (xTimer, pxHigherPriorityTaskWoken)

BaseType_t xTimerStartFromISR( TimerHandle_t xTimer, BaseType_t *pxHigherPriorityTaskWoken );

A version of xTimerStart() that can be called from an interrupt service routine.

Example usage:

```c
// This scenario assumes xBacklightTimer has already been created. When a
// key is pressed, an LCD back-light is switched on. If 5 seconds pass
// without a key being pressed, then the LCD back-light is switched off. In
// this case, the timer is a one-shot timer, and unlike the example given for
// the xTimerReset() function, the key press event handler is an interrupt
// service routine.

// The callback function assigned to the one-shot timer. In this case the
// parameter is not used.
void vBacklightTimerCallback( TimerHandle_t pxTimer )
{
    // The timer expired, therefore 5 seconds must have passed since a key
    // was pressed. Switch off the LCD back-light.
    vSetBacklightState( BACKLIGHT_OFF );
}
```

(续上页)
The key press interrupt service routine.

```c
void vKeyPressEventInterruptHandler( void )
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // Ensure the LCD back-light is on, then restart the timer that is
    // responsible for turning the back-light off after 5 seconds of
    // key inactivity. This is an interrupt service routine so can only
    // call FreeRTOS API functions that end in "FromISR".
    vSetBacklightState( BACKLIGHT_ON );

    // xTimerStartFromISR() or xTimerResetFromISR() could be called here
    // as both cause the timer to re-calculate its expiry time.
    // xHigherPriorityTaskWoken was initialised to pdFALSE when it was
    // declared (in this function).
    if( xTimerStartFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) != pdPASS )
    {
        // The start command was not executed successfully. Take appropriate
        // action here.
    }

    // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
    // should be performed. The syntax required to perform a context switch
    // from inside an ISR varies from port to port, and from compiler to
    // compiler. Inspect the demos for the port you are using to find the
    // actual syntax required.
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // Call the interrupt safe yield function here (actual function
        // depends on the FreeRTOS port being used).
    }
}
```

参数

- `xTimer` - The handle of the timer being started/restarted.
- `pxHigherPriorityTaskWoken` - The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerStartFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerStartFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then `*pxHigherPriorityTaskWoken` will get set to pdTRUE internally within the xTimerStartFromISR() function. If xTimerStartFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

返回 pdFAIL will be returned if the start command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStartFromISR() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

```c
BaseType_t xTimerStopFromISR( TimerHandle_t xTimer, BaseType_t *pxHigherPriorityTaskWoken );
```
A version of xTimerStop() that can be called from an interrupt service routine.

Example usage:

```c
/* // This scenario assumes xTimer has already been created and started. When
  * // an interrupt occurs, the timer should be simply stopped.
  * 
  * // The interrupt service routine that stops the timer.
  * void vAnExampleInterruptServiceRoutine( void )
  * {
  *   BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  *   
  *     // The interrupt has occurred - simply stop the timer.
  *     xHigherPriorityTaskWoken was set to pdFALSE where it was defined
  *     // (within this function). As this is an interrupt service routine, only
  *     // FreeRTOS API functions that end in "FromISR" can be used.
  *     if( xTimerStopFromISR( xTimer, &xHigherPriorityTaskWoken ) != pdPASS )
  *       {
  *         // The stop command was not executed successfully. Take appropriate
  *         // action here.
  *       }
  *     
  *     // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
  *     // should be performed. The syntax required to perform a context switch
  *     // from inside an ISR varies from port to port, and from compiler to
  *     // compiler. Inspect the demos for the port you are using to find the
  *     // actual syntax required.
  *     if( xHigherPriorityTaskWoken != pdFALSE )
  *       {
  *         // Call the interrupt safe yield function here (actual function
  *         // depends on the FreeRTOS port being used).
  *       }
  *   }
  */
```
Example usage:

```c
* // This scenario assumes xTimer has already been created and started. When
* // an interrupt occurs, the period of xTimer should be changed to 500ms.
* // The interrupt service routine that changes the period of xTimer.
* void vAnExampleInterruptServiceRoutine( void )
* {
*   BaseType_t xHigherPriorityTaskWoken = pdFALSE;
*   // The interrupt has occurred - change the period of xTimer to 500ms.
*   xHigherPriorityTaskWoken was set to pdFALSE where it was defined
*   // (within this function). As this is an interrupt service routine, only
*   // FreeRTOS API functions that end in "FromISR" can be used.
*   if( xTimerChangePeriodFromISR( xTimer, &xHigherPriorityTaskWoken ) != pdPASS )
*     {
*       // The command to change the timers period was not executed
*       // successfully. Take appropriate action here.
*       }
*       // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
*       // should be performed. The syntax required to perform a context switch
*       // from inside an ISR varies from port to port, and from compiler to
*       // compiler. Inspect the demos for the port you are using to find the
*       // actual syntax required.
*       if( xHigherPriorityTaskWoken != pdFALSE )
*         {
*           // Call the interrupt safe yield function here (actual function
*           // depends on the FreeRTOS port being used).
*           }
*   }
```
A version of `xTimerReset()` that can be called from an interrupt service routine.

Example usage:

```c
void vBacklightTimerCallback( TimerHandle_t pxTimer )
{
    // The timer expired, therefore 5 seconds must have passed since a key was pressed. Switch off the LCD back-light.
    vSetBacklightState( BACKLIGHT_OFF );
}
```

```c
void vKeyPressEventInterruptHandler( void )
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    // Ensure the LCD back-light is on, then reset the timer that is responsible for turning the back-light off after 5 seconds of key inactivity. This is an interrupt service routine so can only call FreeRTOS API functions that end in "FromISR".
    vSetBacklightState( BACKLIGHT_ON );
    // xTimerStartFromISR() or xTimerResetFromISR() could be called here as both cause the timer to re-calculate its expiry time. xHigherPriorityTaskWoken was initialised to pdFALSE when it was declared (in this function).
    if( xTimerResetFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) != pdPASS )
    {
        // The reset command was not executed successfully. Take appropriate action here.
    }
    // Perform the rest of the key processing here.
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // Call the interrupt safe yield function here (actual function depends on the FreeRTOS port being used).
    }
}
```

**Parameters**

- `xTimer` - The handle of the timer that is to be started, reset, or restarted.
- `pxHigherPriorityTaskWoken` - The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue.
Calling `xTimerResetFromISR()` writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling `xTimerResetFromISR()` causes the timer service/daemon task to leave the Blocked state, and the timer service/daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then `pxHigherPriorityTaskWoken` will get set to `pdTRUE` internally within the `xTimerResetFromISR()` function. If `xTimerResetFromISR()` sets this value to `pdTRUE` then a context switch should be performed before the interrupt exits.

Returns: `pdFAIL` will be returned if the reset command could not be sent to the timer command queue. `pdPASS` will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when `xTimerResetFromISR()` is actually called. The timer service/daemon task priority is set by the `configTIMER_TASK_PRIORITY` configuration constant.

**Type Definitions**

typedef struct tmrTimerControl *TimerHandle_t

typedef void (*TimerCallbackFunction_t)(TimerHandle_t xTimer)

typedef void (*PendedFunction_t)(void*, uint32_t)

**Event Group API**

**Header File**

- `components/freertos/FreeRTOS-Kernel/include/freertos/event_groups.h`

**Functions**

`EventGroupHandle_t xEventGroupCreate(void)`

Create a new event group.

Internally, within the FreeRTOS implementation, event groups use a [small] block of memory, in which the event group’s structure is stored. If an event group is created using `xEventGroupCreate()` then the required memory is automatically dynamically allocated inside the `xEventGroupCreate()` function. (see [https://www.FreeRTOS.org/a00111.html](https://www.FreeRTOS.org/a00111.html)). If an event group is created using `xEventGroupCreateStatic()` then the application writer must instead provide the memory that will get used by the event group. `xEventGroupCreateStatic()` therefore allows an event group to be created without using any dynamic memory allocation.

Although event groups are not related to ticks, for internal implementation reasons the number of bits available for use in an event group is dependent on the `configUSE_16_BIT_TICKS` setting in `FreeRTOSConfig.h`. If `configUSE_16_BIT_TICKS` is 1 then each event group contains 8 usable bits (bit 0 to bit 7). If `configUSE_16_BIT_TICKS` is set to 0 then each event group has 24 usable bits (bit 0 to bit 23). The `EventBits_t` type is used to store event bits within an event group.

**Example usage:**

```c
// Declare a variable to hold the created event group.
EventGroupHandle_t xCreatedEventGroup;

// Attempt to create the event group.
xCreatedEventGroup = xEventGroupCreate();

// Was the event group created successfully?
if( xCreatedEventGroup == NULL )
```

(下頁敘述)
If the event group was created then a handle to the event group is returned. If there was insufficient FreeRTOS heap available to create the event group then NULL is returned. See https://www.FreeRTOS.org/a00111.html

EventGroupHandle_t xEventGroupCreateStatic (StaticEventGroup_t *pxEventGroupBuffer)

Create a new event group.

Internally, within the FreeRTOS implementation, event groups use a small block of memory, in which the event group’s structure is stored. If an event groups is created using xEventGroupCreate() then the required memory is automatically dynamically allocated inside the xEventGroupCreate() function. (see https://www.FreeRTOS.org/a00111.html). If an event group is created using xEventGroupCreateStatic() then the application writer must instead provide the memory that will get used by the event group. xEventGroupCreateStatic() therefore allows an event group to be created without using any dynamic memory allocation.

Although event groups are not related to ticks, for internal implementation reasons the number of bits available for use in an event group is dependent on the configUSE_16_BIT_TICKS setting in FreeRTOSConfig.h. If configUSE_16_BIT_TICKS is 1 then each event group contains 8 usable bits (bit 0 to bit 7). If configUSE_16_BIT_TICKS is set to 0 then each event group has 24 usable bits (bit 0 to bit 23). The EventBits_t type is used to store event bits within an event group.

Example usage:

```c
// StaticEventGroup_t is a publicly accessible structure that has the same
// size and alignment requirements as the real event group structure. It is
// provided as a mechanism for applications to know the size of the event
// group (which is dependent on the architecture and configuration file
// settings) without breaking the strict data hiding policy by exposing the
// real event group internals. This StaticEventGroup_t variable is passed
// into the xSemaphoreCreateEventGroupStatic() function and is used to store
// the event group’s data structures
StaticEventGroup_t xEventGroupBuffer;

// Create the event group without dynamically allocating any memory.
EventGroupHandle_t xEventGroup = xEventGroupCreateStatic( &xEventGroupBuffer );
```

参数 pxEventGroupBuffer – pxEventGroupBuffer must point to a variable of type StaticEventGroup_t, which will be then be used to hold the event group’s data structures, removing the need for the memory to be allocated dynamically.

返回 If the event group was created then a handle to the event group is returned. If pxEventGroupBuffer was NULL then NULL is returned.

EventBits_t xEventGroupWaitBits ( EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToWaitFor, const BaseType_t xClearOnExit, const BaseType_t xWaitForAllBits, TickType_t xTicksToWait )

[Potentially] block to wait for one or more bits to be set within a previously created event group.

This function cannot be called from an interrupt.
Example usage:

```c
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
    EventBits_t uxBits;
    const TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;

    // Wait a maximum of 100ms for either bit 0 or bit 4 to be set within
    // the event group. Clear the bits before exiting.
    uxBits = xEventGroupWaitBits(
        xEventGroup, // The event group being tested.
        BIT_0 | BIT_4, // The bits within the event group to wait for.
        pdTRUE, // BIT_0 and BIT_4 should be cleared before returning.
        pdFALSE, // Don't wait for both bits, either bit will do.
        xTicksToWait ); // Wait a maximum of 100ms for either bit to be set.

    if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
    {
        // xEventGroupWaitBits() returned because both bits were set.
    }
    else if( ( uxBits & BIT_0 ) != 0 )
    {
        // xEventGroupWaitBits() returned because just BIT_0 was set.
    }
    else if( ( uxBits & BIT_4 ) != 0 )
    {
        // xEventGroupWaitBits() returned because just BIT_4 was set.
    }
    else
    {
        // xEventGroupWaitBits() returned because xTicksToWait ticks passed
        // without either BIT_0 or BIT_4 becoming set.
    }
}
```

参数:

- **xEventGroup** — Event group in which the bits are being tested. The event group must have previously been created using a call to xEventGroupCreate().
- **uxBitsToWaitFor** — A bitwise value that indicates the bit or bits to test inside the event group. For example, to wait for bit 0 and/or bit 2 set uxBitsToWaitFor to 0x05. To wait for bits 0 and/or bit 1 and/or bit 2 set uxBitsToWaitFor to 0x07. Etc.
- **xClearOnExit** — If xClearOnExit is set to pdTRUE then any bits within uxBitsToWaitFor that are set within the event group will be cleared before xEventGroupWaitBits() returns if the wait condition was met (if the function returns for a reason other than a timeout). If xClearOnExit is set to pdFALSE then the bits set in the event group are not altered when the call to xEventGroupWaitBits() returns.
- **xWaitForAllBits** — If xWaitForAllBits is set to pdTRUE then xEventGroupWaitBits() will return when either all the bits in uxBitsToWaitFor are set or the specified block time expires. If xWaitForAllBits is set to pdFALSE then xEventGroupWaitBits() will return when any one of the bits set in uxBitsToWaitFor is set or the specified block time expires. The block time is specified by the xTicksToWait parameter.
- **xTicksToWait** — The maximum amount of time (specified in ‘ticks’) to wait for one/all (depending on the xWaitForAllBits value) of the bits specified by uxBitsToWaitFor to become set.
The value of the event group at the time either the bits being waited for became set, or the block time expired. Test the return value to know which bits were set. If `xEventGroupWaitBits()` returned because its timeout expired then not all the bits being waited for will be set. If `xEventGroupWaitBits()` returned because the bits it was waiting for were set then the returned value is the event group value before any bits were automatically cleared in the case that `xClearOnExit` parameter was set to `pdTRUE`.

```c
EventBits_t xEventGroupClearBits (EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToClear)
```
Clear bits within an event group. This function cannot be called from an interrupt.

Example usage:

```c
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

void aFunction ( EventGroupHandle_t xEventGroup )
{
    EventBits_t uxBits;
    // Clear bit 0 and bit 4 in xEventGroup.
    uxBits = xEventGroupClearBits( xEventGroup, // The event group being updated.
                                   BIT_0 | BIT_4 ); // The bits being cleared.
    if(( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ))
    {
        // Both bit 0 and bit 4 were set before xEventGroupClearBits() was called. Both will now be clear (not set).
    }
    else if( uxBits & BIT_0 ) != 0 )
    {
        // Bit 0 was set before xEventGroupClearBits() was called. It will now be clear.
    }
    else if( uxBits & BIT_4 ) != 0 )
    {
        // Bit 4 was set before xEventGroupClearBits() was called. It will now be clear.
    }
    else
    {
        // Neither bit 0 nor bit 4 were set in the first place.
    }
}
```

参数
- `xEventGroup` - The event group in which the bits are to be cleared.
- `uxBitsToClear` - A bitwise value that indicates the bit or bits to clear in the event group. For example, to clear bit 3 only, set `uxBitsToClear` to `0x08`. To clear bit 3 and bit 0 set `uxBitsToClear` to `0x09`.

返回 The value of the event group before the specified bits were cleared.

```c
EventBits_t xEventGroupSetBits (EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToSet)
```
Set bits within an event group. This function cannot be called from an interrupt. `xEVENTGroupSetBitsFromISR()` is a version that can be called from an interrupt.

Setting bits in an event group will automatically unblock tasks that are blocked waiting for the bits.

Example usage:
```c
#define BIT_0 (1 << 0)
#define BIT_4 (1 << 4)

void aFunction(EventGroupHandle_t xEventGroup)
{
    EventBits_t uxBits;
    // Set bit 0 and bit 4 in xEventGroup.
    uxBits = xEventGroupSetBits(
        xEventGroup, // The event group being updated.
        BIT_0 | BIT_4); // The bits being set.
    if( (uxBits & (BIT_0 | BIT_4)) == (BIT_0 | BIT_4) )
    {
        // Both bit 0 and bit 4 remained set when the function returned.
    }
    else if( (uxBits & BIT_0) != 0 )
    {
        // Bit 0 remained set when the function returned, but bit 4 was
        // cleared. It might be that bit 4 was cleared automatically as a
        // task that was waiting for bit 4 was removed from the Blocked
        // state.
    }
    else if( (uxBits & BIT_4) != 0 )
    {
        // Bit 4 remained set when the function returned, but bit 0 was
        // cleared. It might be that bit 0 was cleared automatically as a
        // task that was waiting for bit 0 was removed from the Blocked
        // state.
    }
    else
    {
        // Neither bit 0 nor bit 4 remained set. It might be that a task
        // was waiting for both of the bits to be set, and the bits were
        // cleared as the task left the Blocked state.
    }
}
```

### 参数
- **xEventGroup** — The event group in which the bits are to be set.
- **uxBitsToSet** — A bitwise value that indicates the bit or bits to set. For example, to set bit 0, set `uxBitsToSet` to `0x01`. To set bit 0 and bit 1, set `uxBitsToSet` to `0x03`.

### 返回
The value of the event group at the time the call to `xEventGroupSetBits()` returns. There are two reasons why the returned value might have the bits specified by the `uxBitsToSet` parameter cleared. First, if setting a bit results in a task that was waiting for the bit leaving the blocked state then it is possible the bit will be cleared automatically (see the `xClearOnExit` parameter of `xEventGroupWaitBits()`). Second, any unblocked (or otherwise Ready state) task that has a priority above that of the task that called `xEventGroupSetBits()` will execute and may change the event group value before the call to `xEventGroupSetBits()` returns.

```c
EventBits_t xEventGroupSync(EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToSet, const EventBits_t uxBitsToWaitFor, TickType_t xTicksToWait)
```

Atomically set bits within an event group, then wait for a combination of bits to be set within the same event group. This functionality is typically used to synchronise multiple tasks, where each task has to wait for the other tasks to reach a synchronisation point before proceeding.

This function cannot be used from an interrupt.

The function will return before its block time expires if the bits specified by the `uxBitsToWait` parameter are set, or become set within that time. In this case all the bits specified by `uxBitsToWait` will be automatically cleared before the function returns.
Example usage:

```c
// Bits used by the three tasks.
#define TASK_0_BIT    ( 1 << 0 )
#define TASK_1_BIT    ( 1 << 1 )
#define TASK_2_BIT    ( 1 << 2 )
#define ALL_SYNC_BITS ( TASK_0_BIT | TASK_1_BIT | TASK_2_BIT )

// Use an event group to synchronise three tasks. It is assumed this event
// group has already been created elsewhere.
EventGroupHandle_t xEventBits;

void vTask0( void *pvParameters )
{
    EventBits_t uxReturn;
    TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;
    for( ;; )
    {
        // Perform task functionality here.

        // Set bit 0 in the event flag to note this task has reached the
        // sync point. The other two tasks will set the other two bits defined
        // by ALL_SYNC_BITS. All three tasks have reached the synchronisation
        // point when all the ALL_SYNC_BITS are set. Wait a maximum of 100ms
        // for this to happen.
        uxReturn = xEventGroupSync( xEventBits, TASK_0_BIT, ALL_SYNC_BITS,
                                 xTicksToWait );

        if( ( uxReturn & ALL_SYNC_BITS ) == ALL_SYNC_BITS )
        {
            // All three tasks reached the synchronisation point before the call
            // to xEventGroupSync() timed out.
        }
    }
}

void vTask1( void *pvParameters )
{
    for( ;; )
    {
        // Perform task functionality here.

        // Set bit 1 in the event flag to note this task has reached the
        // synchronisation point. The other two tasks will set the other two
        // bits defined by ALL_SYNC_BITS. All three tasks have reached the
        // synchronisation point when all the ALL_SYNC_BITS are set. Wait
        // indefinitely for this to happen.
        xEventGroupSync( xEventBits, TASK_1_BIT, ALL_SYNC_BITS, portMAX_DELAY );

        // xEventGroupSync() was called with an indefinite block time, so
        // this task will only reach here if the synchronisation was made by all
        // three tasks, so there is no need to test the return value.
    }
}

void vTask2( void *pvParameters )
{
    for( ;; )
    {

    }
```

// Perform task functionality here.

// Set bit 2 in the event flag to note this task has reached the
// synchronisation point. The other two tasks will set the other two
// bits defined by ALL_SYNC_BITS. All three tasks have reached the
// synchronisation point when all the ALL_SYNC_BITS are set. Wait
// indefinitely for this to happen.
xEventGroupSync(xEventBits, TASK_2_BIT, ALL_SYNC_BITS, portMAX_DELAY);

// xEventGroupSync() was called with an indefinite block time, so
// this task will only reach here if the synchronisation was made by all
// three tasks, so there is no need to test the return value.

Parameters:
- **xEventGroup** – The event group in which the bits are being tested. The event group must have previously been created using a call to xEventGroupCreate().
- **uxBitsToSet** – The bits to set in the event group before determining if, and possibly waiting for, all the bits specified by the uxBitsToWait parameter are set.
- **uxBitsToWaitFor** – A bitwise value that indicates the bit or bits to test inside the event group. For example, to wait for bit 0 and bit 2 set uxBitsToWaitFor to 0x05. To wait for bits 0 and bit 1 and bit 2 set uxBitsToWaitFor to 0x07. Etc.
- **xTicksToWait** – The maximum amount of time (specified in ‘ticks’) to wait for all of the bits specified by uxBitsToWaitFor to become set.

Return: The value of the event group at the time either the bits being waited for became set, or the block time expired. Test the return value to know which bits were set. If xEventGroupSync() returned because its timeout expired then not all the bits being waited for will be set. If xEventGroupSync() returned because all the bits it was waiting for were set then the returned value is the event group value before any bits were automatically cleared.

```c
EventBits_t xEventGroupGetBitsFromISR(EventGroupHandle_t xEventGroup)
```

A version of xEventGroupGetBits() that can be called from an ISR.

```c
void vEventGroupDelete(EventGroupHandle_t xEventGroup)
```

Delete an event group that was previously created by a call to xEventGroupCreate(). Tasks that are blocked on the event group will be unblocked and obtain 0 as the event group’s value.

Macros:
```c
xEventGroupClearBitsFromISR(xEventGroup, uxBitsToClear)
```

A version of xEventGroupClearBits() that can be called from an interrupt.

Setting bits in an event group is not a deterministic operation because there are an unknown number of tasks that may be waiting for the bit or bits being set. FreeRTOS does not allow nondeterministic operations to be performed while interrupts are disabled, so protects event groups that are accessed from tasks by suspending the scheduler rather than disabling interrupts. As a result event groups cannot be accessed directly from an interrupt service routine. Therefore xEventGroupClearBitsFromISR() sends a message to the timer task to have the clear operation performed in the context of the timer task.

Example usage:
```c
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

// An event group which it is assumed has already been created by a call to
// xEventGroupCreate().
EventGroupHandle_t xEventGroup;

void anInterruptHandler( void )
{
    // Clear bit 0 and bit 4 in xEventGroup.
    xResult = xEventGroupClearBitsFromISR(
        xEventGroup,
        // The event group being updated.
        BIT_0 | BIT_4 );// The bits being set.

    if( xResult == pdPASS )
    {
        // The message was posted successfully.
    }
}
```

### Parameters

- **xEventGroup** - The event group in which the bits are to be cleared.
- **uxBitsToClear** – A bitwise value that indicates the bit or bits to clear. For example, to clear bit 3 only, set `uxBitsToClear` to 0x08. To clear bit 3 and bit 0 set `uxBitsToClear` to 0x09.

#### Retuns

If the request to execute the function was posted successfully then `pdPASS` is returned, otherwise `pdFALSE` is returned. `pdFALSE` will be returned if the timer service queue was full.

### Function: `xEventGroupSetBitsFromISR` (xEventGroup, uxBitsToSet, pxHigherPriorityTaskWoken)

A version of `xEventGroupSetBits()` that can be called from an interrupt.

Setting bits in an event group is not a deterministic operation because there are an unknown number of tasks that may be waiting for the bit or bits being set. FreeRTOS does not allow nondeterministic operations to be performed in interrupts or from critical sections. Therefore `xEventGroupSetBitsFromISR()` sends a message to the timer task to have the set operation performed in the context of the timer task - where a scheduler lock is used in place of a critical section.

#### Example usage:

```c
#define BIT_0 ( 1 << 0 )
#define BIT_4 ( 1 << 4 )

// An event group which it is assumed has already been created by a call to
// xEventGroupCreate().
EventGroupHandle_t xEventGroup;

void anInterruptHandler( void )
{
    BaseType_t xHigherPriorityTaskWoken, xResult;

    // xHigherPriorityTaskWoken must be initialised to pdFALSE.
    xHigherPriorityTaskWoken = pdFALSE;

    // Set bit 0 and bit 4 in xEventGroup.
    xResult = xEventGroupSetBitsFromISR(
        xEventGroup,
        // The event group being updated.
        BIT_0 | BIT_4 ); // The bits being set.

    &xHigherPriorityTaskWoken
```
参数
- xEventGroup – The event group in which the bits are to be set.
- uxBitsToSet – A bitwise value that indicates the bit or bits to set. For example, to set bit 3 only, set uxBitsToSet to 0x08. To set bit 3 and bit 0 set uxBitsToSet to 0x09.
- pxHigherPriorityTaskWoken – As mentioned above, calling this function will result in a message being sent to the timer daemon task. If the priority of the timer daemon task is higher than the priority of the currently running task (the task the interrupt interrupted) then *pxHigherPriorityTaskWoken will be set to pdTRUE by xEventGroupSetBitsFromISR(), indicating that a context switch should be requested before the interrupt exits. For that reason *pxHigherPriorityTaskWoken must be initialised to pdFALSE. See the example code below.

返回 If the request to execute the function was posted successfully then pdPASS is returned, otherwise pdFALSE is returned. pdFALSE will be returned if the timer service queue was full.

xEventGroupGetBits (xEventGroup)

Returns the current value of the bits in an event group. This function cannot be used from an interrupt.

参数
- xEventGroup – The event group being queried.

返回 The event group bits at the time xEventGroupGetBits() was called.

Type Definitions
typedef struct EventGroupDef_t *EventGroupHandle_t
typedef TickType_t EventBits_t

Stream Buffer API

Header File
- components/freertos/FreeRTOS-Kernel/include/freertos/stream_buffer.h

Functions
size_t xStreamBufferSend (StreamBufferHandle_t xStreamBuffer, const void *pvTxData, size_t xDataLengthBytes, TickType_t xTicksToWait)

Sends bytes to a stream buffer. The bytes are copied into the stream buffer.

: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xStreamBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then
the application writer must place each call to a reading API function (such as xStreamBufferReceive()) inside a critical section and set the receive block time to 0.

Use xStreamBufferSend() to write to a stream buffer from a task. Use xStreamBufferSendFromISR() to write to a stream buffer from an interrupt service routine (ISR).

Example use:

```c
void vAFunction( StreamBufferHandle_t xStreamBuffer )
{
size_t xBytesSent;
uint8_t ucArrayToSend[] = { 0, 1, 2, 3 };
char *pcStringToSend = "String to send";
const TickType_t x100ms = pdMS_TO_TICKS( 100 );

// Send an array to the stream buffer, blocking for a maximum of 100ms to
// wait for enough space to be available in the stream buffer.
xBytesSent = xStreamBufferSend( xStreamBuffer, ( void * ) ucArrayToSend, ...
           sizeof( ucArrayToSend ), x100ms );
if( xBytesSent != sizeof( ucArrayToSend ) )
{
  // The call to xStreamBufferSend() times out before there was enough
  // space in the buffer for the data to be written, but it did
  // successfully write xBytesSent bytes.
}

// Send the string to the stream buffer. Return immediately if there is not
// enough space in the buffer.
xBytesSent = xStreamBufferSend( xStreamBuffer, ( void * ) pcStringToSend, ...
           strlen( pcStringToSend ), 0 );
if( xBytesSent != strlen( pcStringToSend ) )
{
  // The entire string could not be added to the stream buffer because
  // there was not enough free space in the buffer, but xBytesSent bytes
  // were sent. Could try again to send the remaining bytes.
}
}
```

**参数**
- **xStreamBuffer** – The handle of the stream buffer to which a stream is being sent.
- **pvTxData** – A pointer to the buffer that holds the bytes to be copied into the stream buffer.
- **xDataLengthBytes** – The maximum number of bytes to copy from pvTxData into the stream buffer.
- **xTicksToWait** – The maximum amount of time the task should remain in the Blocked state to wait for enough space to become available in the stream buffer, should the stream buffer contain too little space to hold the another xDataLengthBytes bytes. The block time is specified in tick periods, so the absolute time it represents is dependent on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time specified in milliseconds into a time specified in ticks. Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out), provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h. If a task times out before it can write all xDataLengthBytes into the buffer it will still write as many bytes as possible. A task does not use any CPU time when it is in the blocked state.
- **pxHigherPriorityTaskWoken** – The number of bytes written to the stream buffer. If a task times out before it can write all xDataLengthBytes into the buffer it will still write as many bytes as possible.

```c
size_t xStreamBufferSendFromISR( StreamBufferHandle_t xStreamBuffer, const void *pvTxData, size_t xDataLengthBytes, BaseType_t *const pxHigherPriorityTaskWoken)
```
Interrupt safe version of the API function that sends a stream of bytes to the stream buffer.:

Among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xStreamBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xStreamBufferReceive()) inside a critical section and set the receive block time to 0.

Use xStreamBufferSend() to write to a stream buffer from a task. Use xStreamBufferSendFromISR() to write to a stream buffer from an interrupt service routine (ISR).

Example use:

```c
// A stream buffer that has already been created.
StreamBufferHandle_t xStreamBuffer;

void vAnInterruptServiceRoutine( void )
{
    size_t xBytesSent;
    char *pcStringToSend = "String to send";
    BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Initialised to pdFALSE.

    // Attempt to send the string to the stream buffer.
    xBytesSent = xStreamBufferSendFromISR( xStreamBuffer,
        ( void * ) pcStringToSend,
        strlen( pcStringToSend ),
        &xHigherPriorityTaskWoken );

    if( xBytesSent != strlen( pcStringToSend ) )
    {
        // There was not enough free space in the stream buffer for the entire
        // string to be written, ut xBytesSent bytes were written.
    }

    // If xHigherPriorityTaskWoken was set to pdTRUE inside
    // xStreamBufferSendFromISR() then a task that has a priority above the
    // priority of the currently executing task was unblocked and a context
    // switch should be performed to ensure the ISR returns to the unblocked
    // task. In most FreeRTOS ports this is done by simply passing
    // xHigherPriorityTaskWoken into taskYIELD_FROM_ISR(), which will test the
    // variables value, and perform the context switch if necessary. Check the
    // documentation for the port in use for port specific instructions.
    taskYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

**Parameters**
- **xStreamBuffer** – The handle of the stream buffer to which a stream is being sent.
- **pvTxData** – A pointer to the data that is to be copied into the stream buffer.
- **xDataLengthBytes** – The maximum number of bytes to copy from pvTxData into the stream buffer.
- **pxHigherPriorityTaskWoken** – It is possible that a stream buffer will have a task blocked on it waiting for data. Calling xStreamBufferSendFromISR() can make data available, and so cause a task that was waiting for data to leave the Blocked state. If calling xStreamBufferSendFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted),
then, internally, xStreamBufferSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE. If xStreamBufferSendFromISR() sets this value to pdTRUE, then normally a context switch should be performed before the interrupt is exited. This will ensure that the interrupt returns directly to the highest priority Ready state task. *pxHigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function. See the example code below for an example.

The number of bytes actually written to the stream buffer, which will be less than xDataLengthBytes if the stream buffer didn’t have enough free space for all the bytes to be written.

```c
size_t xStreamBufferReceive (StreamBufferHandle_t xStreamBuffer, void *pvRxData, size_t xBufferLengthBytes, TickType_t xTicksToWait)

Receives bytes from a stream buffer.

: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xStreamBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xStreamBufferReceive()) inside a critical section and set the receive block time to 0.

Use xStreamBufferReceive() to read from a stream buffer from a task. Use xStreamBufferReceiveFromISR() to read from a stream buffer from an interrupt service routine (ISR).

Example use:

```c
void vAFunction( StreamBuffer_t xStreamBuffer )
{
    uint8_t ucRxData[ 20 ];
    size_t xReceivedBytes;
    const TickType_t xBlockTime = pdMS_TO_TICKS( 20 );

    // Receive up to another sizeof( ucRxData ) bytes from the stream buffer.
    // Wait in the Blocked state (so not using any CPU processing time) for a
    // maximum of 100ms for the full sizeof( ucRxData ) number of bytes to be
    // available.
    xReceivedBytes = xStreamBufferReceive( xStreamBuffer,
                      ( void * ) ucRxData,
                      sizeof( ucRxData ),
                      xBlockTime );

    if( xReceivedBytes > 0 )
    {
        // A ucRxData contains another xRecievedBytes bytes of data, which can
        // be processed here....
    }
}
```

参数

- **xStreamBuffer** - The handle of the stream buffer from which bytes are to be received.
- **pvRxData** - A pointer to the buffer into which the received bytes will be copied.
- **xBufferLengthBytes** - The length of the buffer pointed to by the pvRxData parameter. This sets the maximum number of bytes to receive in one call. xStreamBufferReceive will return as many bytes as possible up to a maximum set by xBufferLengthBytes.
- **xTicksToWait** - The maximum amount of time the task should remain in the Blocked state to wait for data to become available if the stream buffer is empty. xStreamBufferReceive() will return immediately if xTicksToWait is zero. The block time is specified in...
tick periods, so the absolute time it represents is dependent on the tick frequency. The macro \texttt{pdMS\_TO\_TICKS()} can be used to convert a time specified in milliseconds into a time specified in ticks. Setting \texttt{xTicksToWait} to \texttt{portMAX\_DELAY} will cause the task to wait indefinitely (without timing out), provided \texttt{INCLUDE\_vTaskSuspend} is set to 1 in \texttt{FreeRTOSConfig.h}. A task does not use any CPU time when it is in the Blocked state.

The number of bytes actually read from the stream buffer, which will be less than \texttt{xBufferLengthBytes} if the call to \texttt{xStreamBufferReceive()} timed out before \texttt{xBufferLengthBytes} were available.

\begin{verbatim}
size_t xStreamBufferReceiveFromISR(StreamBufferHandle_t xStreamBuffer, void *pvRxData, size_t xBufferLengthBytes, BaseType_t *const pxHigherPriorityTaskWoken)
\end{verbatim}

An interrupt safe version of the API function that receives bytes from a stream buffer.

Use \texttt{xStreamBufferReceive()} to read bytes from a stream buffer from a task. Use \texttt{xStreamBufferReceiveFromISR()} to read bytes from a stream buffer from an interrupt service routine (ISR).

Example use:

```c
// A stream buffer that has already been created.
StreamBuffer_t xStreamBuffer;

void vAnInterruptServiceRoutine( void )
{
  uint8_t ucRxData[ 20 ];
  size_t xReceivedBytes;
  BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Initialised to pdFALSE.

  // Receive the next stream from the stream buffer.
  xReceivedBytes = xStreamBufferReceiveFromISR( xStreamBuffer,
                                               ( void * ) ucRxData,
                                               sizeof( ucRxData ),
                                               &xHigherPriorityTaskWoken );

  if( xReceivedBytes > 0 )
  {
    // ucRxData contains xReceivedBytes read from the stream buffer.
    // Process the stream here....
  }

  // If xHigherPriorityTaskWoken was set to pdTRUE inside
  // xStreamBufferReceiveFromISR() then a task that has a priority above the
  // priority of the currently executing task was unblocked and a context
  // switch should be performed to ensure the ISR returns to the unblocked
  // task. In most FreeRTOS ports this is done by simply passing
  // xHigherPriorityTaskWoken into taskYIELD\_FROM\_ISR(), which will test the
  // variables value, and perform the context switch if necessary. Check the
  // documentation for the port in use for port specific instructions.
  taskYIELD\_FROM\_ISR( xHigherPriorityTaskWoken );
}
```

参数

- \texttt{xStreamBuffer} – The handle of the stream buffer from which a stream is being received.
- \texttt{pvRxData} – A pointer to the buffer into which the received bytes are copied.
- \texttt{xBufferLengthBytes} – The length of the buffer pointed to by the \texttt{pvRxData} parameter. This sets the maximum number of bytes to receive in one call. \texttt{xStreamBufferReceive()} will return as many bytes as possible up to a maximum set by \texttt{xBufferLengthBytes}.
- \texttt{pxHigherPriorityTaskWoken} – It is possible that a stream buffer will have a task blocked on it waiting for space to become available. Calling \texttt{xStreamBufferReceive-}
FromISR() can make space available, and so cause a task that is waiting for space to leave the Blocked state. If calling xStreamBufferReceiveFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted), then, internally, xStreamBufferReceiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE. If xStreamBufferReceiveFromISR() sets this value to pdTRUE, then normally a context switch should be performed before the interrupt is exited. That will ensure the interrupt returns directly to the highest priority Ready state task. *pxHigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function. See the code example below for an example.

void vStreamBufferDelete (StreamBufferHandle_t xStreamBuffer)

Deletes a stream buffer that was previously created using a call to xStreamBufferCreate() or xStreamBufferCreateStatic(). If the stream buffer was created using dynamic memory (that is, by xStreamBufferCreate()), then the allocated memory is freed.

A stream buffer handle must not be used after the stream buffer has been deleted.

参数 xStreamBuffer – The handle of the stream buffer to be deleted.

BaseType_t xStreamBufferIsFull (StreamBufferHandle_t xStreamBuffer)

Queries a stream buffer to see if it is full. A stream buffer is full if it does not have any free space, and therefore cannot accept any more data.

参数 xStreamBuffer – The handle of the stream buffer being queried.

返回 If the stream buffer is full then pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t xStreamBufferIsEmpty (StreamBufferHandle_t xStreamBuffer)

Queries a stream buffer to see if it is empty. A stream buffer is empty if it does not contain any data.

参数 xStreamBuffer – The handle of the stream buffer being queried.

返回 If the stream buffer is empty then pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t xStreamBufferReset (StreamBufferHandle_t xStreamBuffer)

Resets a stream buffer to its initial, empty, state. Any data that was in the stream buffer is discarded. A stream buffer can only be reset if there are no tasks blocked waiting to either send to or receive from the stream buffer.

参数 xStreamBuffer – The handle of the stream buffer being reset.

返回 If the stream buffer is reset then pdPASS is returned. If there was a task blocked waiting to send to or read from the stream buffer then the stream buffer is not reset and pdFAIL is returned.

size_t xStreamBufferSpacesAvailable (StreamBufferHandle_t xStreamBuffer)

Queries a stream buffer to see how much free space it contains, which is equal to the amount of data that can be sent to the stream buffer before it is full.

参数 xStreamBuffer – The handle of the stream buffer being queried.

返回 The number of bytes that can be written to the stream buffer before the stream buffer would be full.

size_t xStreamBufferBytesAvailable (StreamBufferHandle_t xStreamBuffer)

Queries a stream buffer to see how much data it contains, which is equal to the number of bytes that can be read from the stream buffer before the stream buffer would be empty.

参数 xStreamBuffer – The handle of the stream buffer being queried.

返回 The number of bytes that can be read from the stream buffer before the stream buffer would be empty.

BaseType_t xStreamBufferSetTriggerLevel (StreamBufferHandle_t xStreamBuffer, size_t xTriggerLevel)

A stream buffer’s trigger level is the number of bytes that must be in the stream buffer before a task that is blocked on the stream buffer to wait for data is moved out of the blocked state. For example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 1 then the task will be unblocked when a single
byte is written to the buffer or the task’s block time expires. As another example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 10 then the task will not be unblocked until the stream buffer contains at least 10 bytes or the task’s block time expires. If a reading task’s block time expires before the trigger level is reached then the task will still receive however many bytes are actually available. Setting a trigger level of 0 will result in a trigger level of 1 being used. It is not valid to specify a trigger level that is greater than the buffer size.

A trigger level is set when the stream buffer is created, and can be modified using xStreamBufferSetTriggerLevel().

参数
- xStreamBuffer – The handle of the stream buffer being updated.
- xTriggerLevel – The new trigger level for the stream buffer.

返回 If xTriggerLevel was less than or equal to the stream buffer’s length then the trigger level will be updated and pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t xStreamBufferSendCompletedFromISR(StreamBufferHandle_t xStreamBuffer, BaseType_t pxHigherPriorityTaskWoken)

For advanced users only.

The sbSEND_COMPLETED() macro is called from within the FreeRTOS APIs when data is sent to a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbSEND_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xStreamBufferSendCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbSEND_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

参数
- xStreamBuffer – The handle of the stream buffer to which data was written.
- pxHigherPriorityTaskWoken – pxHigherPriorityTaskWoken should be initialised to pdFALSE before it is passed into xStreamBufferSendCompletedFromISR(). If calling xStreamBufferSendCompletedFromISR() removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then *pxHigherPriorityTaskWoken will get set to pdTRUE indicating that a context switch should be performed before exiting the ISR.

返回 If a task was removed from the Blocked state then pdTRUE is returned. Otherwise pdFALSE is returned.

BaseType_t xStreamBufferReceiveCompletedFromISR(StreamBufferHandle_t xStreamBuffer, BaseType_t pxHigherPriorityTaskWoken)

For advanced users only.

The sbRECEIVE_COMPLETED() macro is called from within the FreeRTOS APIs when data is read out of a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbRECEIVE_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xStreamBufferReceiveCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbRECEIVE_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

参数
- xStreamBuffer – The handle of the stream buffer from which data was read.
- pxHigherPriorityTaskWoken – pxHigherPriorityTaskWoken should be initialised to pdFALSE before it is passed into xStreamBufferReceiveCompletedFromISR(). If calling xStreamBufferReceiveCompletedFromISR() removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then *pxHigherPriorityTaskWoken will get set to pdTRUE indicating that a context switch should be performed before exiting the ISR.

返回 If a task was removed from the Blocked state then pdTRUE is returned. Otherwise pdFALSE is returned.
Macros

**xStreamBufferCreate** (xBuffersizeBytes, xTriggerLevelBytes)

Creates a new stream buffer using dynamically allocated memory. See xStreamBufferCreateStatic() for a version that uses statically allocated memory (memory that is allocated at compile time).

 configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 or left undefined in FreeRTOSConfig.h for xStreamBufferCreate() to be available.

Example use:

```c
void vAFunction( void )
{
    StreamBufferHandle_t xStreamBuffer;
    const size_t xStreamBufferSizeBytes = 100, xTriggerLevel = 10;

    // Create a stream buffer that can hold 100 bytes. The memory used to hold
    // both the stream buffer structure and the data in the stream buffer is
    // allocated dynamically.
    xStreamBuffer = xStreamBufferCreate( xStreamBufferSizeBytes, xTriggerLevel );

    if( xStreamBuffer == NULL )
    {
        // There was not enough heap memory space available to create the
        // stream buffer.
    }
    else
    {
        // The stream buffer was created successfully and can now be used.
    }
}
```

参数

- **xBuffersizeBytes** – The total number of bytes the stream buffer will be able to hold at any one time.
- **xTriggerLevelBytes** – The number of bytes that must be in the stream buffer before a task that is blocked on the stream buffer to wait for data is moved out of the blocked state. For example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 1 then the task will be unblocked when a single byte is written to the buffer or the task’s block time expires. As another example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 10 then the task will not be unblocked until the stream buffer contains at least 10 bytes or the task’s block time expires. If a reading task’s block time expires before the trigger level is reached then the task will still receive however many bytes are actually available. Setting a trigger level of 0 will result in a trigger level of 1 being used. It is not valid to specify a trigger level that is greater than the buffer size.

返回 If NULL is returned, then the stream buffer cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the stream buffer data structures and storage area. A non-NULL value being returned indicates that the stream buffer has been created successfully - the returned value should be stored as the handle to the created stream buffer.

**xStreamBufferCreateStatic** (xBuffersizeBytes, xTriggerLevelBytes, pucStreamBufferStorageArea, pxStaticStreamBuffer)

Creates a new stream buffer using statically allocated memory. See xStreamBufferCreate() for a version that uses dynamically allocated memory.

 configSUPPORT_STATIC_ALLOCATION must be set to 1 in FreeRTOSConfig.h for xStreamBufferCreateStatic() to be available.

Example use:
// Used to dimension the array used to hold the streams. The available space // will actually be one less than this, so 999.
#define STORAGE_SIZE_BYTES 1000

// Defines the memory that will actually hold the streams within the stream // buffer.
static uint8_t ucStorageBuffer[ STORAGE_SIZE_BYTES ];

// The variable used to hold the stream buffer structure.
StaticStreamBuffer_t xStreamBufferStruct;

void MyFunction( void )
{
    StreamBufferHandle_t xStreamBuffer;
    size_t xTriggerLevel = 1;

    xStreamBuffer = xStreamBufferCreateStatic( sizeof( ucBufferStorage ),
                                             xTriggerLevel,
                                             ucBufferStorage,
                                             &xStreamBufferStruct );

    // As neither the pucStreamBufferStorageArea or pxStaticStreamBuffer
    // parameters were NULL, xStreamBuffer will not be NULL, and can be used to
    // reference the created stream buffer in other stream buffer API calls.

    // Other code that uses the stream buffer can go here.
}

参数

- **xBUFFER SIZE BYTES** — The size, in bytes, of the buffer pointed to by the pucStreamBufferStorageArea parameter.
- **xTRIGGER LEVEL BYTES** — The number of bytes that must be in the stream buffer before a task that is blocked on the stream buffer to wait for data is moved out of the blocked state. For example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 1 then the task will be unblocked when a single byte is written to the buffer or the task’s block time expires. As another example, if a task is blocked on a read of an empty stream buffer that has a trigger level of 10 then the task will not be unblocked until the stream buffer contains at least 10 bytes or the task’s block time expires. If a reading task’s block time expires before the trigger level is reached then the task will still receive however many bytes are actually available. Setting a trigger level of 0 will result in a trigger level of 1 being used. It is not valid to specify a trigger level that is greater than the buffer size.
- **pucSTREAM BUFFER STORAGE AREA** — Must point to a uint8_t array that is at least xBUFFER SIZE BYTES + 1 big. This is the array to which streams are copied when they are written to the stream buffer.
- **pxSTATIC STREAM BUFFER** — Must point to a variable of type StaticStreamBuffer_t, which will be used to hold the stream buffer’s data structure.

返回 If the stream buffer is created successfully then a handle to the created stream buffer is returned. If either pucStreamBufferStorageArea or pxStaticstreamBuffer are NULL then NULL is returned.

Type Definitions
typedef struct StreamBufferDef_t *StreamBufferHandle_t

Message Buffer API

Header File
Macros

**xMessageBufferCreate** (xBufferSizeBytes)

Creates a new message buffer using dynamically allocated memory. See **xMessageBufferCreateStatic()** for a version that uses statically allocated memory (memory that is allocated at compile time).

configSUPPORT_DYNAMIC_ALLOCATION must be set to 1 or left undefined in FreeRTOSConfig.h for **xMessageBufferCreate()** to be available.

Example use:

```c
void vAFunction( void )
{
    MessageBufferHandle_t xMessageBuffer;
    const size_t xMessageBufferSizeBytes = 100;

    // Create a message buffer that can hold 100 bytes. The memory used to hold
    // both the message buffer structure and the messages themselves is allocated
    // dynamically. Each message added to the buffer consumes an additional 4
    // bytes which are used to hold the length of the message.
    xMessageBuffer = xMessageBufferCreate( xMessageBufferSizeBytes );

    if( xMessageBuffer == NULL )
    {
        // There was not enough heap memory space available to create the
        // message buffer.
    }
    else
    {
        // The message buffer was created successfully and can now be used.
    }
}
```

**参数**

- **xBufferSizeBytes** - The total number of bytes (not messages) the message buffer will be able to hold at any one time. When a message is written to the message buffer an additional sizeof( size_t ) bytes are also written to store the message’s length. sizeof( size_t ) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architectures a 10 byte message will take up 14 bytes of message buffer space.

**返回** If NULL is returned, then the message buffer cannot be created because there is insufficient heap memory available for FreeRTOS to allocate the message buffer data structures and storage area. A non-NULL value being returned indicates that the message buffer has been created successfully - the returned value should be stored as the handle to the created message buffer.

**xMessageBufferCreateStatic** (xBufferSizeBytes, pucMessageBufferStorageArea, pxStaticMessageBuffer)

Creates a new message buffer using statically allocated memory. See **xMessageBufferCreate()** for a version that uses dynamically allocated memory.

Example use:

```c
// Used to dimension the array used to hold the messages. The available space
// will actually be one less than this, so 999.
#define STORAGE_SIZE_BYTES 1000

// Defines the memory that will actually hold the messages within the message
// buffer.
```
static uint8_t ucStorageBuffer[ STORAGE_SIZE_BYTES ];

// The variable used to hold the message buffer structure.
StaticMessageBuffer_t xMessageBufferStruct;

void MyFunction( void )
{
    MessageBufferHandle_t xMessageBuffer;

    xMessageBuffer = xMessageBufferCreateStatic( sizeof( ucBufferStorage ),
                                                ucBufferStorage,
                                                &xMessageBufferStruct );

    // As neither the pucMessageBufferStorageArea or pxStaticMessageBuffer
    // parameters were NULL, xMessageBuffer will not be NULL, and can be used to
    // reference the created message buffer in other message buffer API calls.

    // Other code that uses the message buffer can go here.
}

参数

• xBufferSizeBytes -- The size, in bytes, of the buffer pointed to by the pucMessageBufferStorageArea parameter. When a message is written to the message buffer an additional sizeof(size_t) bytes are also written to store the message’s length. sizeof(size_t) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architecture a 10 byte message will take up 14 bytes of message buffer space. The maximum number of bytes that can be stored in the message buffer is actually (xBufferSizeBytes - 1).

• pucMessageBufferStorageArea -- Must point to a uint8_t array that is at least xBufferSizeBytes + 1 big. This is the array to which messages are copied when they are written to the message buffer.

• pxStaticMessageBuffer -- Must point to a variable of type StaticMessageBuffer_t, which will be used to hold the message buffer’s data structure.

返回

如果消息缓冲器创建成功，则返回一个指向创建的缓冲器的指针。如果pucMessageBufferStorageArea或pxStaticMessageBuffer为NULL，则返回NULL。

xMessageBufferSend(xMessageBuffer, pvTxData, xDataLengthBytes, xTicksToWait)

将一个离散消息发送到消息缓冲器。消息的长度可以是任何能在缓冲器的空闲空间中放入的长度，并且会被放入缓冲器。

: 与其他FreeRTOS对象不同，消息缓冲器的实现（以及消息缓冲器本身）假定只有一个任务或中断会写入缓冲器（发送者），而只有一个任务或中断会从缓冲器读取（接收者）。发送者和接收者可以是不同的任务或中断，但不像其他FreeRTOS对象，它并不保证有多个不同的发送者或多个不同的接收者。如果发送者和接收者都需要在写入API函数（如xMessageBufferSend()）的线程内执行，并且设置发送超时时间为0。同样，如果存在多个不同的发送者或接收者，则每个发送者和接收者都需要在读取API函数（如xMessageBufferRead()）的线程内执行，并且设置接收超时时间为0。

使用xMessageBufferSend()将消息发送到任务。使用xMessageBufferSendFromISR()将消息发送到中断服务程序（ISR）。

Example use:

void vAFunction( MessageBufferHandle_t xMessageBuffer )
{

(下页继续)
```c
size_t xBytesSent;
uint8_t ucArrayToSend[] = { 0, 1, 2, 3 };
char *pcStringToSend = "String to send";
const TickType_t x100ms = pdMS_TO_TICKS( 100 );

// Send an array to the message buffer, blocking for a maximum of 100ms to
// wait for enough space to be available in the message buffer.
xBytesSent = xMessageBufferSend( xMessageBuffer, ( void * ) ucArrayToSend,
                                 sizeof( ucArrayToSend ), x100ms );

if( xBytesSent != sizeof( ucArrayToSend ) )
{
    // The call to xMessageBufferSend() times out before there was enough
    // space in the buffer for the data to be written.
}

// Send the string to the message buffer. Return immediately if there is
// not enough space in the buffer.
xBytesSent = xMessageBufferSend( xMessageBuffer, ( void * ) pcStringToSend,
                                 strlen( pcStringToSend ), 0 );

if( xBytesSent != strlen( pcStringToSend ) )
{
    // The string could not be added to the message buffer because there was
    // not enough free space in the buffer.
}
```

### 参数
- **xMessageBuffer** - The handle of the message buffer to which a message is being sent.
- **pvTxData** - A pointer to the message that is to be copied into the message buffer.
- **xDataLengthBytes** - The length of the message. That is, the number of bytes to copy from pvTxData into the message buffer. When a message is written to the message buffer an additional sizeof( size_t ) bytes are also written to store the message’s length. sizeof( size_t ) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architecture setting xDataLengthBytes to 20 will reduce the free space in the message buffer by 24 bytes (20 bytes of message data and 4 bytes to hold the message length).
- **xTicksToWait** - The maximum amount of time the calling task should remain in the Blocked state to wait for enough space to become available in the message buffer, should the message buffer have insufficient space when xMessageBufferSend() is called. The calling task will never block if xTicksToWait is zero. The block time is specified in tick periods, so the absolute time it represents is dependent on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time specified in milliseconds into a time specified in ticks. Setting xTicksToWait to portMAX_DELAY will cause the task to wait indefinitely (without timing out), provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h. Tasks do not use any CPU time when they are in the Blocked state.

### 返回
The number of bytes written to the message buffer. If the call to xMessageBufferSend() times out before there was enough space to write the message into the message buffer then zero is returned. If the call did not time out then xDataLengthBytes is returned.

### xMessageBufferSendFromISR
(xMessageBuffer, pvTxData, xDataLengthBytes, pxHigherPriorityTaskWoken)

Interrupt safe version of the API function that sends a discrete message to the message buffer. The message can be any length that fits within the buffer’s free space, and is copied into the buffer.

: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not
safe to have multiple different writers or multiple different readers. If there are to be multiple different writers then the application writer must place each call to a writing API function (such as xMessageBufferSend()) inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers then the application writer must place each call to a reading API function (such as xMessageBufferRead()) inside a critical section and set the receive block time to 0.

Use xMessageBufferSend() to write to a message buffer from a task. Use xMessageBufferSendFromISR() to write to a message buffer from an interrupt service routine (ISR).

Example use:

```c
// A message buffer that has already been created.
MessageBufferHandle_t xMessageBuffer;

void vAnInterruptServiceRoutine( void )
{
    size_t xBytesSent;
    char *pcStringToSend = "String to send";
    BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Initialised to pdFALSE.

    // Attempt to send the string to the message buffer.
    xBytesSent = xMessageBufferSendFromISR( xMessageBuffer,
                                           ( void * ) pcStringToSend,
                                           strlen( pcStringToSend ),
                                           &xHigherPriorityTaskWoken );

    if( xBytesSent != strlen( pcStringToSend ) )
    {
        // The string could not be added to the message buffer because there was
        // not enough free space in the buffer.
    }

    // If xHigherPriorityTaskWoken was set to pdTRUE inside
    // xMessageBufferSendFromISR() then a task that has a priority above the
    // priority of the currently executing task was unblocked and a context
    // switch should be performed to ensure the ISR returns to the unblocked
    // task. In most FreeRTOS ports this is done by simply passing
    // xHigherPriorityTaskWoken into portYIELD_FROM_ISR(), which will test the
    // variables value, and perform the context switch if necessary. Check the
    // documentation for the port in use for port specific instructions.
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

**Parameters**

- `xMessageBuffer` - The handle of the message buffer to which a message is being sent.
- `pvTxData` - A pointer to the message that is to be copied into the message buffer.
- `xDataLengthBytes` - The length of the message. That is, the number of bytes to copy from `pvTxData` into the message buffer. When a message is written to the message buffer an additional sizeof(`size_t`) bytes are also written to store the message’s length. sizeof( `size_t`) is typically 4 bytes on a 32-bit architecture, so on most 32-bit architecture setting xDataLengthBytes to 20 will reduce the free space in the message buffer by 24 bytes (20 bytes of message data and 4 bytes to hold the message length).
- `pxHigherPriorityTaskWoken` - It is possible that a message buffer will have a task blocked on it waiting for data. Calling xMessageBufferSendFromISR() can make data available, and so cause a task that was waiting for data to leave the Blocked state. If calling xMessageBufferSendFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted), then, internally, xMessageBufferSendFromISR() will set `*pxHigherPriorityTaskWoken` to `pdTRUE`. If xMessageBufferSendFromISR() sets this value to `pdTRUE`, then normally a context switch should be performed before the interrupt is exited. This
will ensure that the interrupt returns directly to the highest priority Ready state task. *px-
HigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function.
See the code example below for an example.

The number of bytes actually written to the message buffer. If the message buffer didn’t have
enough free space for the message to be stored then 0 is returned, otherwise xDataLengthBytes
is returned.

**xMessageBufferReceive** (xMessageBuffer, pvRxData, xBufferLengthBytes, xTicksToWait)
Receives a discrete message from a message buffer. Messages can be of variable length and are copied out of
the buffer.

- Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implement-
ation, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that
will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It
is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not
safe to have multiple different writers or multiple different readers. If there are to be multiple different writers
then the application writer must place each call to a writing API function (such as xMessageBufferSend())
inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers
then the application writer must place each call to a reading API function (such as xMessageBufferRead())
inside a critical section and set the receive block time to 0.

Use xMessageBufferReceive() to read from a message buffer from a task. Use xMessageBufferReceive-
FromISR() to read from a message buffer from an interrupt service routine (ISR).

**Example use:**

```c
void vAFunction( MessageBuffer_t xMessageBuffer )
{
  uint8_t ucRxData[20];
  size_t xReceivedBytes;
  const TickType_t xBlockTime = pdMS_TO_TICKS(20);

  // Receive the next message from the message buffer. Wait in the Blocked
  // state (so not using any CPU processing time) for a maximum of 100ms for
  // a message to become available.
  xReceivedBytes = xMessageBufferReceive( xMessageBuffer,
    (void*) ucRxData,
    sizeof( ucRxData ),
    xBlockTime );

  if( xReceivedBytes > 0 )
  {
    // ucRxData contains a message that is xReceivedBytes long. Process
    // the message here....
  }
}
```

**参数**
- **xMessageBuffer** – The handle of the message buffer from which a message is being
  received.
- **pvRxData** – A pointer to the buffer into which the received message is to be copied.
- **xBufferLengthBytes** – The length of the buffer pointed to by the pvRxData parameter. This sets the maximum length of the message that can be received. If xBufferLengthBytes is too small to hold the next message then the message will be left in the message buffer and 0 will be returned.
- **xTicksToWait** – The maximum amount of time the task should remain in the Blocked
  state to wait for a message, should the message buffer be empty. xMessageBufferReceive() will return immediately if xTicksToWait is zero and the message buffer is empty. The block time is specified in tick periods, so the absolute time it represents is dependent

---

**Submit Document Feedback**
on the tick frequency. The macro pdMS_TO_TICKS() can be used to convert a time
specified in milliseconds into a time specified in ticks. Setting xTicksToWait to port-
MAX_DELAY will cause the task to wait indefinitely (without timing out), provided IN-
CLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h. Tasks do not use any CPU time
when they are in the Blocked state.

返回 The length, in bytes, of the message read from the message buffer, if any. If xMessageBuf-
ferReceive() times out before a message became available then zero is returned. If the length
of the message is greater than xBufferLengthBytes then the message will be left in the message
buffer and zero is returned.

\textbf{xMessageBufferReceiveFromISR}(xMessageBuffer, pvRxData, xBufferLengthBytes,
pxHigherPriorityTaskWoken)

An interrupt safe version of the API function that receives a discrete message from a message buffer. Messages
can be of variable length and are copied out of the buffer.

: Uniquely among FreeRTOS objects, the stream buffer implementation (so also the message buffer implementa-
tion, as message buffers are built on top of stream buffers) assumes there is only one task or interrupt that
will write to the buffer (the writer), and only one task or interrupt that will read from the buffer (the reader). It
is safe for the writer and reader to be different tasks or interrupts, but, unlike other FreeRTOS objects, it is not
safe to have multiple different writers or multiple different readers. If there are to be multiple different writers
then the application writer must place each call to a writing API function (such as xMessageBufferSend())
inside a critical section and set the send block time to 0. Likewise, if there are to be multiple different readers
then the application writer must place each call to a reading API function (such as xMessageBufferRead())
inside a critical section and set the receive block time to 0.

Use \textbf{xMessageBufferReceive()} to read from a message buffer from a task. Use \textbf{xMessageBufferReceive-
FromISR()} to read from a message buffer from an interrupt service routine (ISR).

Example use:

```c
// A message buffer that has already been created.
MessageBuffer_t xMessageBuffer;

void vAnInterruptServiceRoutine( void )
{
  uint8_t ucRxData[ 20 ];
  size_t xReceivedBytes;
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;  // Initialised to pdFALSE.

  // Receive the next message from the message buffer.
  xReceivedBytes = xMessageBufferReceiveFromISR( xMessageBuffer,
    (void *) ucRxData,
    sizeof( ucRxData ),
    &xHigherPriorityTaskWoken );

  if( xReceivedBytes > 0 )
  {
    // A ucRxData contains a message that is xReceivedBytes long. Process
    // the message here....
  }

  // If xHigherPriorityTaskWoken was set to pdTRUE inside
  // xMessageBufferReceiveFromISR() then a task that has a priority above the
  // priority of the currently executing task was unblocked and a context
  // switch should be performed to ensure the ISR returns to the unblocked
  // task. In most FreeRTOS ports this is done by simply passing
  // xHigherPriorityTaskWoken into portYIELD_FROM_ISR(), which will test the
  // variables value, and perform the context switch if necessary. Check the
  // documentation for the port in use for port specific instructions.
```

(下页继续)
PortYIELD_FROM_ISR(xHigherPriorityTaskWoken);

参数

- **xMessageBuffer** - The handle of the message buffer from which a message is being received.
- **pvRxData** - A pointer to the buffer into which the received message is to be copied.
- **xBufferLengthBytes** - The length of the buffer pointed to by the pvRxData parameter. This sets the maximum length of the message that can be received. If xBufferLengthBytes is too small to hold the next message then the message will be left in the message buffer and 0 will be returned.
- **pxHigherPriorityTaskWoken** - It is possible that a message buffer will have a task blocked on it waiting for space to become available. Calling xMessageBufferReceiveFromISR() can make space available, and so cause a task that is waiting for space to leave the Blocked state. If calling xMessageBufferReceiveFromISR() causes a task to leave the Blocked state, and the unblocked task has a priority higher than the currently executing task (the task that was interrupted), then, internally, xMessageBufferReceiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE. If xMessageBufferReceiveFromISR() sets this value to pdTRUE, then normally a context switch should be performed before the interrupt is exited. That will ensure the interrupt returns directly to the highest priority Ready state task. *pxHigherPriorityTaskWoken should be set to pdFALSE before it is passed into the function. See the code example below for an example.

返回 The length, in bytes, of the message read from the message buffer, if any.

vMessageBufferDelete(xMessageBuffer)

Delete a message buffer that was previously created using a call to xMessageBufferCreate() or xMessageBufferCreateStatic(). If the message buffer was created using dynamic memory (that is, by xMessageBufferCreate()), then the allocated memory is freed.

A message buffer handle must not be used after the message buffer has been deleted.

参数

- **xMessageBuffer** - The handle of the message buffer to be deleted.

xMessageBufferIsFull(xMessageBuffer)

Tests to see if a message buffer is full. A message buffer is full if it cannot accept any more messages, of any size, until space is made available by a message being removed from the message buffer.

参数

- **xMessageBuffer** - The handle of the message buffer being queried.

返回 If the message buffer referenced by xMessageBuffer is full then pdTRUE is returned. Otherwise pdFALSE is returned.

xMessageBufferIsEmpty(xMessageBuffer)

Tests to see if a message buffer is empty (does not contain any messages).

参数

- **xMessageBuffer** - The handle of the message buffer being queried.

返回 If the message buffer referenced by xMessageBuffer is empty then pdTRUE is returned. Otherwise pdFALSE is returned.

xMessageBufferReset(xMessageBuffer)

Resets a message buffer to its initial empty state, discarding any message it contained.

A message buffer can only be reset if there are no tasks blocked on it.

参数

- **xMessageBuffer** - The handle of the message buffer being reset.
If the message buffer was reset then pdPASS is returned. If the message buffer could not be reset because either there was a task blocked on the message queue to wait for space to become available, or to wait for a a message to be available, then pdFAIL is returned.

**xMessageBufferSpaceAvailable** *(xMessageBuffer)*

Returns the number of bytes of free space in the message buffer.

**参数**

- **xMessageBuffer** – The handle of the message buffer being queried.

**返回** The number of bytes that can be written to the message buffer before the message buffer would be full. When a message is written to the message buffer an additional sizeof( size_t ) bytes are also written to store the message’s length. sizeof( size_t ) is typically 4 bytes on a 32-bit architecture, so if xMessageBufferSpacesAvailable() returns 10, then the size of the largest message that can be written to the message buffer is 6 bytes.

**xMessageBufferNextLengthBytes** *(xMessageBuffer)*

Returns the length (in bytes) of the next message in a message buffer. Useful if xMessageBufferReceive() returned 0 because the size of the buffer passed into xMessageBufferReceive() was too small to hold the next message.

**参数**

- **xMessageBuffer** – The handle of the message buffer being queried.

**返回** The length (in bytes) of the next message in the message buffer, or 0 if the message buffer is empty.

**xMessageBufferSendCompletedFromISR** *(xMessageBuffer, pxHigherPriorityTaskWoken)*

For advanced users only.

The sbSEND_COMPLETED() macro is called from within the FreeRTOS APIs when data is sent to a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbSEND_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xMessageBufferSendCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbSEND_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

**参数**

- **xMessageBuffer** – The handle of the stream buffer to which data was written.
- **pxHigherPriorityTaskWoken** – *pxHigherPriorityTaskWoken should be initialised to pdFALSE before it is passed into xMessageBufferSendCompletedFromISR(). If calling xMessageBufferSendCompletedFromISR() removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then *px-HigherPriorityTaskWoken will get set to pdTRUE indicating that a context switch should be performed before exiting the ISR.

**返回** If a task was removed from the Blocked state then pdTRUE is returned. Otherwise pdFALSE is returned.

**xMessageBufferReceiveCompletedFromISR** *(xMessageBuffer, pxHigherPriorityTaskWoken)*

For advanced users only.

The sbRECEIVE_COMPLETED() macro is called from within the FreeRTOS APIs when data is read out of a message buffer or stream buffer. If there was a task that was blocked on the message or stream buffer waiting for data to arrive then the sbRECEIVE_COMPLETED() macro sends a notification to the task to remove it from the Blocked state. xMessageBufferReceiveCompletedFromISR() does the same thing. It is provided to enable application writers to implement their own version of sbRECEIVE_COMPLETED(), and MUST NOT BE USED AT ANY OTHER TIME.

See the example implemented in FreeRTOS/Demo/Minimal/MessageBufferAMP.c for additional information.

**参数**
Chapter 2. API

- `xMessageBuffer` - The handle of the stream buffer from which data was read.
- `pxHigherPriorityTaskWoken` - `pxHigherPriorityTaskWoken` should be initialised to `pdFALSE` before it is passed into `xMessageBufferReceiveCompletedFromISR()`. If calling `xMessageBufferReceiveCompletedFromISR()` removes a task from the Blocked state, and the task has a priority above the priority of the currently running task, then `*pxHigherPriorityTaskWoken` will get set to `pdTRUE` indicating that a context switch should be performed before exiting the ISR.

If a task was removed from the Blocked state then `pdTRUE` is returned. Otherwise `pdFALSE` is returned.

Type Definitions

typedef void *MessageBufferHandle_t

Type by which message buffers are referenced. For example, a call to `xMessageBufferCreate()` returns an `MessageBufferHandle_t` variable that can then be used as a parameter to `xMessageBufferSend()`, `xMessageBufferReceive()`, etc.

2.10.11 FreeRTOS Supplemental Features

ESP-IDF uses a modified version of FreeRTOS v10.4.3 that contains significant changes for SMP compatibility (see [ESP-IDF FreeRTOS SMP Changes](#)). However, in addition to ESP-IDF FreeRTOS, various features are also provided by ESP-IDF to supplement the features offered by FreeRTOS.

This document describes these supplemental features added to ESP-IDF. This document is split into the following sections:

### Contents

- **FreeRTOS Supplemental Features**
  - Overview
  - Ring Buffers
  - ESP-IDF Tick and Idle Hooks
  - TLSP Deletion Callbacks
  - Component Specific Properties
  - API Reference

### Overview

ESP-IDF FreeRTOS is modified version of based on the Xtensa port of FreeRTOS v10.4.3 with significant modifications for SMP compatibility (see [ESP-IDF FreeRTOS SMP Changes](#)). However, various new features specific to ESP-IDF FreeRTOS have been added. The features are as follows:

- **Ring buffers**: Ring buffers provide a FIFO buffer that can accept entries of arbitrary lengths.
- **ESP-IDF Tick and Idle Hooks**: ESP-IDF provides multiple custom tick interrupt hooks and idle task hooks that are more numerous and more flexible when compared to FreeRTOS tick and idle hooks.
- **Thread Local Storage Pointer (TLSP) Deletion Callbacks**: TLSP Deletion callbacks are run automatically when a task is deleted, thus allowing users to clean up their TLSPs automatically.
- **Component Specific Properties**: Currently added only one component specific property `ORIG_INCLUDE_PATH`.

### Ring Buffers

The ESP-IDF FreeRTOS ring buffer is a strictly FIFO buffer that supports arbitrarily sized items. Ring buffers are a more memory efficient alternative to FreeRTOS queues in situations where the size of items is variable. The capacity
of a ring buffer is not measured by the number of items it can store, but rather by the amount of memory used for storing items. The ring buffer provides API to send an item, or to allocate space for an item in the ring buffer to be filled manually by the user. For efficiency reasons, items are always retrieved from the ring buffer by reference. As a result, all retrieved items must also be returned to the ring buffer by using \texttt{vRingbufferReturnItem()} or \texttt{vRingbufferReturnItemFromISR()}, in order for them to be removed from the ring buffer completely. The ring buffers are split into the three following types:

\textbf{No-Split buffers} will guarantee that an item is stored in contiguous memory and will not attempt to split an item under any circumstances. Use No-Split buffers when items must occupy contiguous memory. \textit{Only this buffer type allows you to get the data item address and write to the item by yourself.} Refer the documentation of the functions \texttt{xRingbufferSendAcquire()} and \texttt{xRingbufferSendComplete()} for more details.

\textbf{Allow-Split buffers} will allow an item to be split in two parts when wrapping around the end of the buffer if there is enough space at the tail and the head of the buffer combined to store the item. Allow-Split buffers are more memory efficient than No-Split buffers but can return an item in two parts when retrieving.

\textbf{Byte buffers} do not store data as separate items. All data is stored as a sequence of bytes, and any number of bytes can be sent or retrieved each time. Use byte buffers when separate items do not need to be maintained (e.g. a byte stream).

\textbf{备注:} No-Split buffers and Allow-Split buffers will always store items at 32-bit aligned addresses. Therefore, when retrieving an item, the item pointer is guaranteed to be 32-bit aligned. This is useful especially when you need to send some data to the DMA.

\textbf{备注:} Each item stored in No-Split or Allow-Split buffers will require an additional 8 bytes for a header. Item sizes will also be rounded up to a 32-bit aligned size (multiple of 4 bytes), however the true item size is recorded within the header. The sizes of No-Split and Allow-Split buffers will also be rounded up when created.

\textbf{Usage} The following example demonstrates the usage of \texttt{xRingbufferCreate()} and \texttt{xRingbufferSend()} to create a ring buffer and then send an item to it.

```
#include "freertos/ringbuf.h"
static char tx_item[] = "test_item";
...

//Create ring buffer
RingbufHandle_t buf_handle;
buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
if (buf_handle == NULL) {
    printf("Failed to create ring buffer\n");
}

//Send an item
UBaseType_t res = xRingbufferSend(buf_handle, tx_item, sizeof(tx_item), pdMS_TO_TICKS(1000));
if (res != pdTRUE) {
    printf("Failed to send item\n");
}
```

The following example demonstrates the usage of \texttt{xRingbufferSendAcquire()} and \texttt{xRingbufferSendComplete()} instead of \texttt{xRingbufferSend()} to acquire memory on the ring buffer (of type \texttt{RINGBUF_TYPE_NOSPLIT}) and then send an item to it. This adds one more step, but allows getting the address of the memory to write to, and writing to the memory yourself.

```
#include "freertos/ringbuf.h"
#include "soc/lldesc.h"
```
typedef struct {
    llldesc_t dma_desc;
    uint8_t buf[1];
} dma_item_t;

#define DMA_ITEM_SIZE(N) (sizeof(lldesc_t)+(((N)+3)&(~3)))

...  //Retrieve space for DMA descriptor and corresponding data buffer
    //This has to be done with SendAcquire, or the address may be different when
    we copy
    dma_item_t item;
    UBaseType_t res = xRingbufferSendAcquire(buf_handle, &item, DMA_ITEM_SIZE(buffer_size), pdMS_TO_TICKS(1000));
    if (res != pdTRUE) {
        printf("Failed to acquire memory for item\n");
    }
    item->dma_desc = (lldesc_t) {
        .size = buffer_size,
        .length = buffer_size,
        .eof = 0,
        .owner = 1,
        .buf = &item->buf,
    };
    //Actually send to the ring buffer for consumer to use
    res = xRingbufferSendComplete(buf_handle, &item);
    if (res != pdTRUE) {
        printf("Failed to send item\n");
    }

The following example demonstrates retrieving and returning an item from a No-Split ring buffer using
xRingbufferReceive() and vRingbufferReturnItem()

    ...  //Receive an item from no-split ring buffer
    size_t item_size;
    char *item = (char *)xRingbufferReceive(buf_handle, &item_size, pdMS_TO_TICKS(1000));

    //Check received item
    if (item != NULL) {
        //Print item
        for (int i = 0; i < item_size; i++) {
            printf("%c", item[i]);
        }
        printf("\n");
        //Return Item
        vRingbufferReturnItem(buf_handle, (void *)item);
    } else {
        //Failed to receive item
        printf("Failed to receive item\n");
    }

The following example demonstrates retrieving and returning an item from an Allow-Split ring buffer using
xRingbufferReceiveSplit() and vRingbufferReturnItem()
The following example demonstrates retrieving and returning an item from a byte buffer using \texttt{xRingbufferReceiveUpTo()} and \texttt{vRingbufferReturnItem()}

For ISR safe versions of the functions used above, call \texttt{xRingbufferSendFromISR()}, \texttt{xRingbufferReceiveFromISR()}, \texttt{xRingbufferReceiveSplitFromISR()}, \texttt{xRingbufferReceiveUpToFromISR()}, and \texttt{vRingbufferReturnItemFromISR()}

\begin{verbatim}
//Receive data from byte buffer
size_t item_size;
char *item = (char *)xRingbufferReceiveUpTo(buf_handle, &item_size, pdMS_TO_TICKS(1000), sizeof(tx_item));

//Check received data
if (item != NULL) {
    //Print item
    for (int i = 0; i < item_size; i++) {
        printf("%c", item[i]);
    }
    printf(\"\n\n\return Item\n\n\fromISR()\\
\end{verbatim}

\textbf{备注}：Two calls to RingbufferReceive[UpTo][FromISR]() are required if the bytes wraps around the end of the ring buffer.

\textbf{Sending to Ring Buffer}  The following diagrams illustrate the differences between No-Split and Allow-Split buffers as compared to byte buffers with regard to sending items/data. The diagrams assume that three items of sizes 18, 3, and 27 bytes are sent respectively to a buffer of 128 bytes.
For No-Split and Allow-Split buffers, a header of 8 bytes precedes every data item. Furthermore, the space occupied by each item is rounded up to the nearest 32-bit aligned size in order to maintain overall 32-bit alignment. However, the true size of the item is recorded inside the header which will be returned when the item is retrieved.

Referring to the diagram above, the 18, 3, and 27 byte items are rounded up to 20, 4, and 28 bytes respectively. An 8 byte header is then added in front of each item.

Byte buffers treat data as a sequence of bytes and does not incur any overhead (no headers). As a result, all data sent to a byte buffer is merged into a single item.

Referring to the diagram above, the 18, 3, and 27 byte items are sequentially written to the byte buffer and merged into a single item of 48 bytes.

Using SendAcquire and SendComplete Items in No-Split buffers are acquired (by SendAcquire) in strict FIFO order and must be sent to the buffer by SendComplete for the data to be accessible by the consumer. Multiple items can be sent or acquired without calling SendComplete, and the items do not necessarily need to be completed in the order they were acquired. However, the receiving of data items must occur in FIFO order, therefore not calling SendComplete for the earliest acquired item will prevent the subsequent items from being received.

The following diagrams illustrate what will happen when SendAcquire and SendComplete don’t happen in the same order. At the beginning, there is already a data item of 16 bytes sent to the ring buffer. Then SendAcquire is called to acquire space of 20, 8, 24 bytes on the ring buffer.

![Diagram](image-url)
After that, we fill (use) the buffers, and send them to the ring buffer by `SendComplete` in the order of 8, 24, 20. When 8 bytes and 24 bytes data are sent, the consumer still can only get the 16 bytes data item. Hence, if `SendComplete` is not called for the 20 bytes, it will not be available, nor will the data items following the 20 bytes item.

When the 20 bytes item is finally completed, all the 3 data items can be received now, in the order of 20, 8, 24 bytes, right after the 16 bytes item existing in the buffer at the beginning.

Allow-Split buffers and byte buffers do not allow using `SendAcquire` or `SendComplete` since acquired buffers are required to be complete (not wrapped).

**Wrap around**  The following diagrams illustrate the differences between No-Split, Allow-Split, and byte buffers when a sent item requires a wrap around. The diagrams assume a buffer of 128 bytes with 56 bytes of free space that wraps around and a sent item of 28 bytes.

![Diagram](image)

**図 33: Wrap around in No-Split buffers**

No-Split buffers will **only store an item in continuous free space and will not split an item under any circumstances**. When the free space at the tail of the buffer is insufficient to completely store the item and its header, the free space at the tail will be **marked as dummy data**. The buffer will then wrap around and store the item in the free space at the head of the buffer.

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to store the 28 byte item. Therefore, the 16 bytes is marked as dummy data and the item is written to the free space at the head of the buffer instead.

![Diagram](image)

**図 34: Wrap around in Allow-Split buffers**

Allow-Split buffers will attempt to **split the item into two parts** when the free space at the tail of the buffer is insufficient to store the item data and its header. Both parts of the split item will have their own headers (therefore incurring an extra 8 bytes of overhead).

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to store the 28 byte item. Therefore, the item is split into two parts (8 and 20 bytes) and written as two parts to the buffer.

**備注:**  Allow-Split buffers treat both parts of the split item as two separate items, therefore call `xRingbufferReceiveSplit()` instead of `xRingbufferReceive()` to receive both parts of a split item in a thread safe manner.
Byte buffers will **store as much data as possible into the free space at the tail of buffer**. The remaining data will then be stored in the free space at the head of the buffer. No overhead is incurred when wrapping around in byte buffers.

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to completely store the 28 bytes of data. Therefore, the 16 bytes of free space is filled with data, and the remaining 12 bytes are written to the free space at the head of the buffer. The buffer now contains data in two separate continuous parts, and each continuous part will be treated as a separate item by the byte buffer.

**Retrieving/Returning** The following diagrams illustrate the differences between No-Split and Allow-Split buffers as compared to byte buffers in retrieving and returning data.

Items in No-Split buffers and Allow-Split buffers are **retrieved in strict FIFO order** and **must be returned** for the occupied space to be freed. Multiple items can be retrieved before returning, and the items do not necessarily need to be returned in the order they were retrieved. However, the freeing of space must occur in FIFO order, therefore not returning the earliest retrieved item will prevent the space of subsequent items from being freed.

Referring to the diagram above, the **16, 20, and 8 byte items are retrieved in FIFO order**. However, the items are not returned in the order they were retrieved. First, the 20 byte item is returned followed by the 8 byte and the 16 byte items. The space is not freed until the first item, i.e., the 16 byte item is returned.

Byte buffers **do not allow multiple retrievals before returning** (every retrieval must be followed by a return before another retrieval is permitted). When using `xRingbufferReceive()` or `xRingbufferReceiveFromISR()`, all continuous stored data will be retrieved. `xRingbufferReceiveUpTo()` or `xRingbufferReceiveUpToFromISR()` can be used to restrict the maximum number of bytes retrieved. Since every retrieval must be followed by a return, the space will be freed as soon as the data is returned.

Referring to the diagram above, the 38 bytes of continuous stored data at the tail of the buffer is retrieved, returned, and freed. The next call to `xRingbufferReceive()` or `xRingbufferReceiveFromISR()` then wraps around and does the same to the 30 bytes of continuous stored data at the head of the buffer.
### Ring Buffers with Queue Sets

Ring buffers can be added to FreeRTOS queue sets using `xRingbufferAddToQueueSetRead()` such that every time a ring buffer receives an item or data, the queue set is notified. Once added to a queue set, every attempt to retrieve an item from a ring buffer should be preceded by a call to `xQueueSelectFromSet()`. To check whether the selected queue set member is the ring buffer, call `xRingbufferCanRead()`.

The following example demonstrates queue set usage with ring buffers.

```c
#include "freertos/queue.h"
#include "freertos/ringbuf.h"
...

// Create ring buffer and queue set
RingbufHandle_t buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
QueueSetHandle_t queue_set = xQueueCreateSet(3);

// Add ring buffer to queue set
if (xRingbufferAddToQueueSetRead(buf_handle, queue_set) != pdTRUE) {
    printf("Failed to add to queue set\n");
}
...

// Block on queue set
QueueSetMemberHandle_t member = xQueueSelectFromSet(queue_set, pdMS_TO_TICKS(1000));

// Check if member is ring buffer
if (member != NULL && xRingbufferCanRead(buf_handle, member) == pdTRUE) {
    // Member is ring buffer, receive item from ring buffer
    size_t item_size;
    char *item = (char *)xRingbufferReceive(buf_handle, &item_size, 0);

    // Handle item
    ...
} else {
    ...
}
```

### Ring Buffers with Static Allocation

The `xRingbufferCreateStatic()` can be used to create ring buffers with specific memory requirements (such as a ring buffer being allocated in external RAM). All blocks of memory used by a ring buffer must be manually allocated beforehand then passed to the `xRingbufferCreateStatic()` to be initialized as a ring buffer. These blocks include the following:

- The ring buffer’s data structure of type `StaticRingbuffer_t`
Chapter 2. API 参考

- The ring buffer’s storage area of size xBufferSize. Note that xBufferSize must be 32-bit aligned for No-Split and Allow-Split buffers.

The manner in which these blocks are allocated will depend on the users requirements (e.g. all blocks being statically declared, or dynamically allocated with specific capabilities such as external RAM).

备注: When deleting a ring buffer created via xRingbufferCreateStatic(), the function vRingbufferDelete() will not free any of the memory blocks. This must be done manually by the user after vRingbufferDelete() is called.

The code snippet below demonstrates a ring buffer being allocated entirely in external RAM.

```c
#include "freertos/ringbuf.h"
#include "freertos/semphr.h"
#include "esp_heap_caps.h"

#define BUFFER_SIZE 400  // 32-bit aligned size
#define BUFFER_TYPE RINGBUF_TYPE_NOSPLIT

... 

//Allocate ring buffer data structure and storage area into external RAM
StaticRingbuffer_t *buffer_struct = (StaticRingbuffer_t *)heap_caps_malloc(sizeof(StaticRingbuffer_t), MALLOC_CAP_SPIRAM);
uint8_t *buffer_storage = (uint8_t *)heap_caps_malloc(sizeof(uint8_t) * BUFFER_SIZE, MALLOC_CAP_SPIRAM);

//Create a ring buffer with manually allocated memory
RingbufHandle_t handle = xRingbufferCreateStatic(BUFFER_SIZE, BUFFER_TYPE, buffer_storage, buffer_struct);

... 

//Delete the ring buffer after used
vRingbufferDelete(handle);

//Manually free all blocks of memory
free(buffer_struct);
free(buffer_storage);
```

**Priority Inversion** Ideally, ring buffers can be used with multiple tasks in an SMP fashion where the highest priority task will always be serviced first. However due to the usage of binary semaphores in the ring buffer’s underlying implementation, priority inversion may occur under very specific circumstances.

The ring buffer governs sending by a binary semaphore which is given whenever space is freed on the ring buffer. The highest priority task waiting to send will wait repeatedly take the semaphore until sufficient free space becomes available or until it times out. Ideally this should prevent any lower priority tasks from being serviced as the semaphore should always be given to the highest priority task.

However, in between iterations of acquiring the semaphore, there is a gap in the critical section which may permit another task (on the other core or with an even higher priority) to free some space on the ring buffer and as a result give the semaphore. Therefore, the semaphore will be given before the highest priority task can re-acquire the semaphore. This will result in the semaphore being acquired by the second-highest priority task waiting to send, hence causing priority inversion.

This side effect will not affect ring buffer performance drastically given if the number of tasks using the ring buffer simultaneously is low, and the ring buffer is not operating near maximum capacity.

**ESP-IDF Tick and Idle Hooks**

FreeRTOS allows applications to provide a tick hook and an idle hook at compile time:
Chapter 2. API

- FreeRTOS tick hook can be enabled via the `CONFIG_FREERTOS_USE_TICK_HOOK` option. The application must provide the `void vApplicationTickHook( void )` callback.
- FreeRTOS idle hook can be enabled via the `CONFIG_FREERTOS_USE_IDLE_HOOK` option. The application must provide the `void vApplicationIdleHook( void )` callback.

However, the FreeRTOS tick hook and idle hook have the following drawbacks:

- The FreeRTOS hooks are registered at compile time
- Only one of each hook can be registered
- On multi-core targets, the FreeRTOS hooks are symmetric, meaning each CPU’s tick interrupt and idle tasks ends up calling the same hook.

Therefore, ESP-IDF tick and idle hooks are provided to supplement the features of FreeRTOS tick and idle hooks. The ESP-IDF hooks have the following features:

- The hooks can be registered and deregistered at run-time
- Multiple hooks can be registered (with a maximum of 8 hooks of each type per CPU)
- On multi-core targets, the hooks can be asymmetric, meaning different hooks can be registered to each CPU

ESP-IDF hooks can be registered and deregistered using the following API:

- For tick hooks:
  - Register using `esp_register_freertos_tick_hook()` or `esp_register_freertos_tick_hook_for_cpu()`.
  - Deregister using `esp_deregister_freertos_tick_hook()` or `esp_deregister_freertos_tick_hook_for_cpu()`.

- For idle hooks:
  - Register using `esp_register_freertos_idle_hook()` or `esp_register_freertos_idle_hook_for_cpu()`.
  - Deregister using `esp_deregister_freertos_idle_hook()` or `esp_deregister_freertos_idle_hook_for_cpu()`.

备注： The tick interrupt stays active while the cache is disabled, therefore any tick hook (FreeRTOS or ESP-IDF) functions must be placed in internal RAM. Please refer to the SPI flash API documentation for more details.

### TLSP Deletion Callbacks

Vanilla FreeRTOS provides a Thread Local Storage Pointers (TLSP) feature. These are pointers stored directly in the Task Control Block (TCB) of a particular task. TLSPs allow each task to have its own unique set of pointers to data structures. Vanilla FreeRTOS expects users to:

- set a task’s TLSPs by calling `vTaskSetThreadLocalStoragePointer()` after the task has been created.
- get a task’s TLSPs by calling `pvTaskGetThreadLocalStoragePointer()` during the task’s lifetime.
- free the memory pointed to by the TLSPs before the task is deleted.

However, there can be instances where users may want the freeing of TLSP memory to be automatic. Therefore, ESP-IDF FreeRTOS provides the additional feature of TLSP deletion callbacks. These user provided deletion callbacks are called automatically when a task is deleted, thus allows the TLSP memory to be cleaned up without needing to add the cleanup logic explicitly to the code of every task.

The TLSP deletion callbacks are set in a similar fashion to the TLSPs themselves.

- `vTaskSetThreadLocalStoragePointerAndDelCallback()` sets both a particular TLSP and its associated callback.
- Calling the Vanilla FreeRTOS function `vTaskSetThreadLocalStoragePointer()` will simply set the TLSP’s associated Deletion Callback to `NULL` meaning that no callback will be called for that TLSP during task deletion.

When implementing TLSP callbacks, users should note the following:
[The callback must never attempt to block or yield] and critical sections should be kept as short as possible.

The callback is called shortly before a deleted task’s memory is freed. Thus, the callback can either be called from `vTaskDelete()` itself, or from the idle task.

**Component Specific Properties**

Besides standard component variables that are available with basic cmake build properties, FreeRTOS component also provides arguments (only one so far) for simpler integration with other modules:

- `ORIG_INCLUDE_PATH` - contains an absolute path to freertos root include folder. Thus instead of `#include "freertos/FreeRTOS.h"` you can refer to headers directly: `#include "FreeRTOS.h"`.

**API Reference**

**Ring Buffer API**

**Header File**

- `components/esp_ringbuf/include/freertos/ringbuf.h`

**Functions**

- `RingbufHandle_t xRingbufferCreate(size_t xBufferSize, RingbufferType_t xBufferType)`
  
  Create a ring buffer.

  备注: `xBufferSize` of no-split/allow-split buffers will be rounded up to the nearest 32-bit aligned size.

  **参数**
  
  - `xBufferSize` - [in] Size of the buffer in bytes. Note that items require space for a header in no-split/allow-split buffers
  - `xBufferType` - [in] Type of ring buffer, see documentation.

  **返回**
  
  A handle to the created ring buffer, or NULL in case of error.

- `RingbufHandle_t xRingbufferCreateNoSplit(size_t xItemSize, size_t xItemNum)`
  
  Create a ring buffer of type `RINGBUF_TYPE_NOSPLIT` for a fixed `item_size`.

  This API is similar to `xRingbufferCreate()`, but it will internally allocate additional space for the headers.

  **参数**
  
  - `xItemSize` - [in] Size of each item to be put into the ring buffer
  - `xItemNum` - [in] Maximum number of items the buffer needs to hold simultaneously

  **返回**
  
  A RingbufHandle_t handle to the created ring buffer, or NULL in case of error.

- `RingbufHandle_t xRingbufferCreateStatic(size_t xBufferSize, RingbufferType_t xBufferType, uint8_t *pucRingbufferStorage, StaticRingbuffer_t *pxStaticRingbuffer)`
  
  Create a ring buffer but manually provide the required memory.

  备注: `xBufferSize` of no-split/allow-split buffers MUST be 32-bit aligned.

  **参数**
  
  - `xBufferSize` - [in] Size of the buffer in bytes.
  - `xBufferType` - [in] Type of ring buffer, see documentation
  - `pucRingbufferStorage` - [in] Pointer to the ring buffer’s storage area. Storage area must have the same size as specified by `xBufferSize`
• `pxStaticRingbuffer` - [in] Pointed to a struct of type `StaticRingbuffer_t` which will be used to hold the ring buffer’s data structure

返回 A handle to the created ring buffer

```c
BaseType_t xRingbufferSend(RingbufHandle_t xRingbuffer, const void *pvItem, size_t xItemSize,
                             TickType_t xTicksToWait)
```

Insert an item into the ring buffer.

Attempt to insert an item into the ring buffer. This function will block until enough free space is available or until it times out.

备注: For no-split/allow-split ring buffers, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

参数
- `xRingbuffer` - [in] Ring buffer to insert the item into
- `pvItem` - [in] Pointer to data to insert. NULL is allowed if `xItemSize` is 0.
- `xItemSize` - [in] Size of data to insert.
- `xTicksToWait` - [in] Ticks to wait for room in the ring buffer.

返回
- `pdTRUE` if succeeded
- `pdFALSE` on time-out or when the data is larger than the maximum permissible size of the buffer

```c
BaseType_t xRingbufferSendFromISR(RingbufHandle_t xRingbuffer, const void *pvItem, size_t xItemSize,
                                    BaseType_t *pxHigherPriorityTaskWoken)
```

Insert an item into the ring buffer in an ISR.

Attempt to insert an item into the ring buffer from an ISR. This function will return immediately if there is insufficient free space in the buffer.

备注: For no-split/allow-split ring buffers, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

参数
- `xRingbuffer` - [in] Ring buffer to insert the item into
- `pvItem` - [in] Pointer to data to insert. NULL is allowed if `xItemSize` is 0.
- `xItemSize` - [in] Size of data to insert.
- `pxHigherPriorityTaskWoken` - [out] Value pointed to will be set to `pdTRUE` if the function woke up a higher priority task.

返回
- `pdTRUE` if succeeded
- `pdFALSE` when the ring buffer does not have space.

```c
BaseType_t xRingbufferSendAcquire(RingbufHandle_t xRingbuffer, void **ppvItem, size_t xItemSize,
                                   TickType_t xTicksToWait)
```

Acquire memory from the ring buffer to be written to by an external source and to be sent later.

Attempt to allocate buffer for an item to be sent into the ring buffer. This function will block until enough free space is available or until it times out.

The item, as well as the following items `SendAcquire` or `Send` after it, will not be able to be read from the ring buffer until this item is actually sent into the ring buffer.

备注: Only applicable for no-split ring buffers now, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.
Chapter 2. API 参考

aligned fashion.

参数
- xRingbuffer [in] Ring buffer to allocate the memory
- ppvItem [out] Double pointer to memory acquired (set to NULL if no memory were retrieved)
- xItemSize [in] Size of item to acquire.
- xTicksToWait [in] Ticks to wait for room in the ring buffer.

返回
- pdTRUE if succeeded
- pdFALSE on time-out or when the data is larger than the maximum permissible size of the buffer

BaseType_t **xRingbufferSendComplete (RingbufHandle_t xRingbuffer, void *pvItem)
Actually send an item into the ring buffer allocated before by xRingbufferSendAcquire.

备注： Only applicable for no-split ring buffers. Only call for items allocated by xRingbufferSendAcquire.

参数
- xRingbuffer [in] Ring buffer to insert the item into
- pvItem [in] Pointer to item in allocated memory to insert.

返回
- pdTRUE if succeeded
- pdFALSE if fail for some reason.

void *xRingbufferReceive (RingbufHandle_t xRingbuffer, size_t *pxItemSize, TickType_t xTicksToWait)
Retrieve an item from the ring buffer.

Attempt to retrieve an item from the ring buffer. This function will block until an item is available or until it times out.

备注： A call to vRingbufferReturnItem() is required after this to free the item retrieved.

参数
- xRingbuffer [in] Ring buffer to retrieve the item from
- pxItemSize [out] Pointer to a variable to which the size of the retrieved item will be written.
- xTicksToWait [in] Ticks to wait for items in the ring buffer.

返回
- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL on timeout, *pxItemSize is untouched in that case.

void *xRingbufferReceiveFromISR (RingbufHandle_t xRingbuffer, size_t *pxItemSize)
Retrieve an item from the ring buffer in an ISR.

Attempt to retrieve an item from the ring buffer. This function returns immediately if there are no items available for retrieval.

备注： A call to vRingbufferReturnItemFromISR() is required after this to free the item retrieved.

备注： Byte buffers do not allow multiple retrievals before returning an item
Chapter 2. API

### Two calls to RingbufferReceiveFromISR() are required if the bytes wrap around the end of the ring buffer.

#### Parameters
- **xRingbuffer** [in] Ring buffer to retrieve the item from
- **pxItemSize** [out] Pointer to a variable to which the size of the retrieved item will be written.

#### Returns
- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL when the ring buffer is empty. *pxItemSize is untouched in that case.

```c
BaseType_t xRingbufferReceiveSplit(RingbufHandle_t xRingbuffer, void **ppvHeadItem, void **ppvTailItem, size_t *pxHeadItemSize, size_t *pxTailItemSize, TickType_t xTicksToWait)
```

Retrieve a split item from an allow-split ring buffer.

Attempt to retrieve a split item from an allow-split ring buffer. If the item is not split, only a single item is retrieved. If the item is split, both parts will be retrieved. This function will block until an item is available or until it times out.

#### Notes
- Call(s) to vRingbufferReturnItem() is required after this to free up the item(s) retrieved.
- This function should only be called on allow-split buffers.

```c
BaseType_t xRingbufferReceiveSplitFromISR(RingbufHandle_t xRingbuffer, void **ppvHeadItem, void **ppvTailItem, size_t *pxHeadItemSize, size_t *pxTailItemSize)
```

Retrieve a split item from an allow-split ring buffer in an ISR.

Attempt to retrieve a split item from an allow-split ring buffer in an ISR. If the item is not split, only a single item is retrieved. If the item is split, both parts will be retrieved. This function returns immediately if there are no items available for retrieval.

#### Notes
- Calls to vRingbufferReturnItemFromISR() is required after this to free up the item(s) retrieved.
- This function should only be called on allow-split buffers.
Chapter 2. API

- **xRingbuffer** - [in] Ring buffer to retrieve the item from
- **ppvHeadItem** - [out] Double pointer to first part (set to NULL if no items were retrieved)
- **ppvTailItem** - [out] Double pointer to second part (set to NULL if item is not split)
- **pxHeadItemSize** - [out] Pointer to size of first part (unmodified if no items were retrieved)
- **pxTailItemSize** - [out] Pointer to size of second part (unmodified if item is not split)

void *xRingbufferReceiveUpTo (RingbufHandle_t xRingbuffer, size_t *pxItemSize, TickType_t xTicksToWait, size_t xMaxSize)

Retrieve bytes from a byte buffer, specifying the maximum amount of bytes to retrieve.

Attempt to retrieve data from a byte buffer whilst specifying a maximum number of bytes to retrieve. This function will block until there is data available for retrieval or until it times out.

**备注:** A call to vRingbufferReturnItem() is required after this to free up the data retrieved.

**备注:** This function should only be called on byte buffers

**备注:** Byte buffers do not allow multiple retrievals before returning an item

**备注:** Two calls to RingbufferReceiveUpTo() are required if the bytes wrap around the end of the ring buffer.

**参数**
- **xRingbuffer** - [in] Ring buffer to retrieve the item from
- **pxItemSize** - [out] Pointer to a variable to which the size of the retrieved item will be written.
- **xTicksToWait** - [in] Ticks to wait for items in the ring buffer.
- **xMaxSize** - [in] Maximum number of bytes to return.

**返回**
- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL on timeout, *pxItemSize is untouched in that case.

void *xRingbufferReceiveUpToFromISR (RingbufHandle_t xRingbuffer, size_t *pxItemSize, size_t xMaxSize)

Retrieve bytes from a byte buffer, specifying the maximum amount of bytes to retrieve. Call this from an ISR.

Attempt to retrieve bytes from a byte buffer whilst specifying a maximum number of bytes to retrieve. This function will return immediately if there is no data available for retrieval.

**备注:** A call to vRingbufferReturnItemFromISR() is required after this to free up the data received.

**备注:** This function should only be called on byte buffers

**备注:** Byte buffers do not allow multiple retrievals before returning an item
### 参数
- **xRingbuffer** - [in] Ring buffer to retrieve the item from
- **pxItemSize** - [out] Pointer to a variable to which the size of the retrieved item will be written.
- **xMaxSize** - [in] Maximum number of bytes to return.

### 返回
- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL when the ring buffer is empty, *pxItemSize is untouched in that case.

```c
void vRingbufferReturnItem(RingbufHandle_t xRingbuffer, void *pvItem)
```

Return a previously-retrieved item to the ring buffer.

**备注:** If a split item is retrieved, both parts should be returned by calling this function twice

### 参数
- **xRingbuffer** - [in] Ring buffer the item was retrieved from
- **pvItem** - [in] Item that was received earlier

```c
void vRingbufferReturnItemFromISR(RingbufHandle_t xRingbuffer, void *pvItem, BaseType_t *pxHigherPriorityTaskWoken)
```

Return a previously-retrieved item to the ring buffer from an ISR.

**备注:** If a split item is retrieved, both parts should be returned by calling this function twice

### 参数
- **xRingbuffer** - [in] Ring buffer the item was retrieved from
- **pvItem** - [in] Item that was received earlier
- **pxHigherPriorityTaskWoken** - [out] Value pointed to will be set to pdTRUE if the function woke up a higher priority task.

```c
void vRingbufferDelete(RingbufHandle_t xRingbuffer)
```

Delete a ring buffer.

**备注:** This function will not deallocate any memory if the ring buffer was created using xRingbufferCreateStatic(). Deallocation must be done manually by the user.

### 参数 **xRingbuffer** - [in] Ring buffer to delete

```c
size_t xRingbufferGetMaxItemSize(RingbufHandle_t xRingbuffer)
```

Get maximum size of an item that can be placed in the ring buffer.

This function returns the maximum size an item can have if it was placed in an empty ring buffer.

**备注:** The max item size for a no-split buffer is limited to ((buffer_size/2)-header_size). This limit is imposed so that an item of max item size can always be sent to an empty no-split buffer regardless of the internal positions of the buffer’s read/write/free pointers.
size_t xRingbufferGetCurFreeSize (RingbufHandle_t xRingbuffer)
Get current free size available for an item/data in the buffer.
This gives the real time free space available for an item/data in the ring buffer. This represents the maximum
size an item/data can have if it was currently sent to the ring buffer.

参数 xRingbuffer -[in] Ring buffer to query
返回 Current free size, in bytes, available for an entry

BaseType_t xRingbufferAddToQueueSetRead (RingbufHandle_t xRingbuffer, QueueHandle_t xQueueSet)
Add the ring buffer’s read semaphore to a queue set.
The ring buffer’s read semaphore indicates that data has been written to the ring buffer. This function adds
the ring buffer’s read semaphore to a queue set.

参数 • xRingbuffer -[in] Ring buffer to add to the queue set
• xQueueSet -[in] Queue set to add the ring buffer’s read semaphore to
返回 • pdTRUE on success, pdFALSE otherwise

BaseType_t xRingbufferCanRead (RingbufHandle_t xRingbuffer, QueueHandle_t xQueueSet)
Check if the selected queue set member is the ring buffer’s read semaphore.
This API checks if queue set member returned from xQueueSelectFromSet() is the read semaphore of this
ring buffer. If so, this indicates the ring buffer has items waiting to be retrieved.

参数 • xRingbuffer -[in] Ring buffer which should be checked
• xMember -[in] Member returned from xQueueSelectFromSet
返回 • pdTRUE when semaphore belongs to ring buffer
• pdFALSE otherwise.

BaseType_t xRingbufferRemoveFromQueueSetRead (RingbufHandle_t xRingbuffer, QueueHandle_t xQueueSet)
Remove the ring buffer’s read semaphore from a queue set.
This specifically removes a ring buffer’s read semaphore from a queue set. The read semaphore is used to
indicate when data has been written to the ring buffer.

参数 • xRingbuffer -[in] Ring buffer to remove from the queue set
• xQueueSet -[in] Queue set to remove the ring buffer’s read semaphore from
返回 • pdTRUE on success
• pdFALSE otherwise

void vRingbufferGetInfo (RingbufHandle_t xRingbuffer, UBaseType_t *uxFree, UBaseType_t *uxRead,
UBaseType_t *uxWrite, UBaseType_t *uxAcquire, UBaseType_t *uxItemsWaiting)
Get information about ring buffer status.

Get information of a ring buffer’s current status such as free/read/write/acquire pointer positions, and number of items waiting to be retrieved. Arguments can be set to NULL if they are not required.

### Arguments

- `xRingbuffer` [in] Ring buffer to remove from the queue set
- `uxFree` [out] Pointer use to store free pointer position
- `uxRead` [out] Pointer use to store read pointer position
- `uxWrite` [out] Pointer use to store write pointer position
- `uxAcquire` [out] Pointer use to store acquire pointer position
- `uxItemsWaiting` [out] Pointer use to store number of items (bytes for byte buffer) waiting to be retrieved

```c
void xRingbufferPrintInfo (RingbufHandle_t xRingbuffer)
```

Debugging function to print the internal pointers in the ring buffer.

### Structures

**struct xSTATIC_RINGBUFFER**

Struct that is equivalent in size to the ring buffer’s data structure.

The contents of this struct are not meant to be used directly. This structure is meant to be used when creating a statically allocated ring buffer where this struct is of the exact size required to store a ring buffer’s control data structure.

### Type Definitions

```c
typedef void *RingbufHandle_t
```

Type by which ring buffers are referenced. For example, a call to `xRingbufferCreate()` returns a `RingbufHandle_t` variable that can then be used as a parameter to `xRingbufferSend()`, `xRingbufferReceive()`, etc.

```c
typedef struct xSTATIC_RINGBUFFER StaticRingbuffer_t
```

Struct that is equivalent in size to the ring buffer’s data structure.

The contents of this struct are not meant to be used directly. This structure is meant to be used when creating a statically allocated ring buffer where this struct is of the exact size required to store a ring buffer’s control data structure.

### Enumerations

```c
enum RingbufferType_t
```

Values:

- **RINGBUF_TYPE_NOSPLIT**
  
  No-split buffers will only store an item in contiguous memory and will never split an item. Each item requires an 8 byte overhead for a header and will always internally occupy a 32-bit aligned size of space.

- **RINGBUF_TYPE_ALLOWSPLIT**
  
  Allow-split buffers will split an item into two parts if necessary in order to store it. Each item requires an 8 byte overhead for a header, splitting incurs an extra header. Each item will always internally occupy a 32-bit aligned size of space.
enumerator **RINGBUF_TYPE_BYTEBUF**

Byte buffers store data as a sequence of bytes and do not maintain separate items, therefore byte buffers have no overhead. All data is stored as a sequence of byte and any number of bytes can be sent or retrieved each time.

enumerator **RINGBUF_TYPE_MAX**

### Hooks API

#### Header File

- `components/esp_system/include/esp_freertos_hooks.h`

#### Functions

**esp_err_t esp_register_freertos_idle_hook_for_cpu** *(esp_freertos_idle_cb_t new_idle_cb, UBaseType_t cpuid)*

Register a callback to be called from the specified core’s idle hook. The callback should return true if it should be called by the idle hook once per interrupt (or FreeRTOS tick), and return false if it should be called repeatedly as fast as possible by the idle hook.

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_idle_cb</td>
<td>Callback to be called</td>
</tr>
<tr>
<td>cpuid</td>
<td>id of the core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_OK</td>
<td>Callback registered to the specified core’s idle hook</td>
</tr>
<tr>
<td>ESP_ERR_NO_MEM</td>
<td>No more space on the specified core’s idle hook to register callback</td>
</tr>
<tr>
<td>ESP_ERR_INVALID_ARG</td>
<td>cpuid is invalid</td>
</tr>
</tbody>
</table>

**esp_err_t esp_register_freertos_idle_hook** *(esp_freertos_idle_cb_t new_idle_cb)*

Register a callback to the idle hook of the core that calls this function. The callback should return true if it should be called by the idle hook once per interrupt (or FreeRTOS tick), and return false if it should be called repeatedly as fast as possible by the idle hook.

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_idle_cb</td>
<td>Callback to be called</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_OK</td>
<td>Callback registered to the calling core’s idle hook</td>
</tr>
<tr>
<td>ESP_ERR_NO_MEM</td>
<td>No more space on the calling core’s idle hook to register callback</td>
</tr>
</tbody>
</table>

**esp_err_t esp_register_freertos_tick_hook_for_cpu** *(esp_freertos_tick_cb_t new_tick_cb, UBaseType_t cpuid)*

Register a callback to be called from the specified core’s tick hook.

<table>
<thead>
<tr>
<th>参数</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_tick_cb</td>
<td>Callback to be called</td>
</tr>
<tr>
<td>cpuid</td>
<td>id of the core</td>
</tr>
</tbody>
</table>

警告: Idle callbacks MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.
**Chapter 2. API 参考**

返回
- ESP_OK: Callback registered to specified core’s tick hook
- ESP_ERR_NO_MEM: No more space on the specified core’s tick hook to register the callback
- ESP_ERR_INVALID_ARG: cpuid is invalid

*esp_err_t* `esp_register_freertos_tick_hook(esp_freertos_tick_cb_t new_tick_cb)`

Register a callback to be called from the calling core’s tick hook.

参数 `new_tick_cb` - [in] Callback to be called

返回
- ESP_OK: Callback registered to the calling core’s tick hook
- ESP_ERR_NO_MEM: No more space on the calling core’s tick hook to register the callback

void `esp_deregister_freertos_idle_hook_for_cpu(esp_freertos_idle_cb_t old_idle_cb, UBaseType_t cpuid)`

Unregister an idle callback from the idle hook of the specified core.

参数
- `old_idle_cb` - [in] Callback to be unregistered
- `cpuid` - [in] id of the core

void `esp_deregister_freertos_idle_hook(esp_freertos_idle_cb_t old_idle_cb)`

Unregister an idle callback. If the idle callback is registered to the idle hooks of both cores, the idle hook will be unregistered from both cores.

参数 `old_idle_cb` - [in] Callback to be unregistered

void `esp_deregister_freertos_tick_hook_for_cpu(esp_freertos_tick_cb_t old_tick_cb, UBaseType_t cpuid)`

Unregister a tick callback from the tick hook of the specified core.

参数
- `old_tick_cb` - [in] Callback to be unregistered
- `cpuid` - [in] id of the core

void `esp_deregister_freertos_tick_hook(esp_freertos_tick_cb_t old_tick_cb)`

Unregister a tick callback. If the tick callback is registered to the tick hooks of both cores, the tick hook will be unregistered from both cores.

参数 `old_tick_cb` - [in] Callback to be unregistered

**Type Definitions**

typedef bool (*`esp_freertos_idle_cb_t`) (void)

typedef void (*`esp_freertos_tick_cb_t`) (void)

**2.10.12 Heap Memory Allocation**

**Stack and Heap**

ESP-IDF applications use the common computer architecture patterns of stack (dynamic memory allocated by program control flow) and heap (dynamic memory allocated by function calls), as well as statically allocated memory (allocated at compile time).

Because ESP-IDF is a multi-threaded RTOS environment, each RTOS task has its own stack. By default, each of these stacks is allocated from the heap when the task is created. (See `xTaskCreateStatic()` for the alternative where stacks are statically allocated.)
Because ESP32 uses multiple types of RAM, it also contains multiple heaps with different capabilities. A capabilities-based memory allocator allows apps to make heap allocations for different purposes.

For most purposes, the standard libc `malloc()` and `free()` functions can be used for heap allocation without any special consideration.

However, in order to fully make use of all of the memory types and their characteristics, ESP-IDF also has a capabilities-based heap memory allocator. If you want to have memory with certain properties (for example, DMA-Capable Memory or executable-memory), you can create an OR-mask of the required capabilities and pass that to `heap_caps_malloc()`.

### Memory Capabilities

The ESP32 contains multiple types of RAM:

- **DRAM** (Data RAM) is memory used to hold data. This is the most common kind of memory accessed as heap.
- **IRAM** (Instruction RAM) usually holds executable data only. If accessed as generic memory, all accesses must be **32-bit aligned**.
- **D/IRAM** is RAM which can be used as either Instruction or Data RAM.

For more details on these internal memory types, see 存储器类型.

It’s also possible to connect external SPI RAM to the ESP32, which can be integrated into the ESP32’s memory map using the flash cache, and accessed similarly to DRAM.

DRAM uses capability **MALLOC_CAP_8BIT** (accessible in single byte reads and writes). To test the free DRAM heap size at runtime, call `heap_caps_get_free_size(MALLOC_CAP_8BIT)`.

When calling `malloc()`, the ESP-IDF `malloc()` implementation internally calls `heap_caps_malloc_default(size)`. This will allocate memory with capability **MALLOC_CAP_DEFAULT**, which is byte-addressable.

Because `malloc` uses the capabilities-based allocation system, memory allocated using `heap_caps_malloc()` can be freed by calling the standard `free()` function.

### Available Heap

**DRAM** At startup, the DRAM heap contains all data memory which is not statically allocated by the app. Reducing statically allocated buffers will increase the amount of available free heap.

To find the amount of statically allocated memory, use the `idf.py size` command.

**IRAM** At startup, the IRAM heap contains all instruction memory which is not used by the app executable code. The `idf.py size` command can be used to find the amount of IRAM used by the app.

**D/IRAM** Some memory in the ESP32 is available as either DRAM or IRAM. If memory is allocated from a D/IRAM region, the free heap size for both types of memory will decrease.
Heap Sizes

At startup, all ESP-IDF apps log a summary of all heap addresses (and sizes) at level Info:

<table>
<thead>
<tr>
<th>Heap Size Details</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>I (252) heap_init: Initializing. RAM available for dynamic allocation:</td>
<td></td>
</tr>
<tr>
<td>I (259) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM</td>
<td></td>
</tr>
<tr>
<td>I (265) heap_init: At 3FFB2EC8 len 0002D138 (180 KiB): DRAM</td>
<td></td>
</tr>
<tr>
<td>I (272) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM</td>
<td></td>
</tr>
<tr>
<td>I (278) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM</td>
<td></td>
</tr>
<tr>
<td>I (284) heap_init: At 4008944C len 0016BB4 (90 KiB): IRAM</td>
<td></td>
</tr>
</tbody>
</table>

Finding available heap

See Heap Information.

Special Capabilities

DMA-Capable Memory

Use the MALLOC_CAP_DMA flag to allocate memory which is suitable for use with hardware DMA engines (for example SPI and I2S). This capability flag excludes any external PSRAM.

32-Bit Accessible Memory

If a certain memory structure is only addressed in 32-bit units, for example an array of ints or pointers, it can be useful to allocate it with the MALLOC_CAP_32BIT flag. This also allows the allocator to give out IRAM memory; something which it can’t do for a normal malloc() call. This can help to use all the available memory in the ESP32.

Please note that on ESP32 series chips, MALLOC_CAP_32BIT cannot be used for storing floating-point variables. This is because MALLOC_CAP_32BIT may return instruction RAM, and the floating-point assembly instructions on ESP32 cannot access instruction RAM.

Memory allocated with MALLOC_CAP_32BIT can only be accessed via 32-bit reads and writes, any other type of access will generate a fatal LoadStoreError exception.

External SPI Memory

When external RAM is enabled, external SPI RAM under 4MiB in size can be allocated using standard malloc calls, or via heap_caps_malloc(MALLOC_CAP_SPIRAM), depending on configuration. See 設置片外 RAM for more details.

To use the region above the 4MiB limit, you can use the himem API.

Thread Safety

Heap functions are thread safe, meaning they can be called from different tasks simultaneously without any limitations.

It is technically possible to call malloc, free, and related functions from interrupt handler (ISR) context (see Calling heap related functions from ISR). However this is not recommended, as heap function calls may delay other interrupts. It is strongly recommended to refactor applications so that any buffers used by an ISR are pre-allocated outside of the ISR. Support for calling heap functions from ISRs may be removed in a future update.

Calling related functions from ISR

The following functions from the heap component can be called form interrupt handler (ISR):

- heap_caps_malloc()
- heap_caps_malloc_default()
- heap_caps_realloc_default()
- heap_caps_malloc_prefer()
- heap_caps_realloc_prefer()
- heap_caps_calloc_prefer()
- heap_caps_free()
- heap_caps_realloc()
- heap_caps_calloc()
Chapter 2. API

- heap_caps_aligned_alloc()
- heap_caps_aligned_free()

Note however this practice is strongly discouraged.

Heap Tracing & Debugging

The following features are documented on the Heap Memory Debugging page:

- Heap Information (free space, etc.)
- Heap Corruption Detection
- Heap Tracing (memory leak detection, monitoring, etc.)

Implementation Notes

Knowledge about the regions of memory in the chip comes from the "soc" component, which contains memory layout information for the chip, and the different capabilities of each region. Each region’s capabilities are prioritised, so that (for example) dedicated DRAM and IRAM regions will be used for allocations ahead of the more versatile D/IRAM regions.

Each contiguous region of memory contains its own memory heap. The heaps are created using the multi_heap functionality. multi_heap allows any contiguous region of memory to be used as a heap.

The heap capabilities allocator uses knowledge of the memory regions to initialize each individual heap. Allocation functions in the heap capabilities API will find the most appropriate heap for the allocation (based on desired capabilities, available space, and preferences for each region’s use) and then calling multi_heap_malloc() for the heap situated in that particular region.

Calling free() involves finding the particular heap corresponding to the freed address, and then calling multi_heap_free() on that particular multi_heap instance.

API Reference - Heap Allocation

Header File

- components/heap/include/esp_heap_caps.h

Functions

esp_err_t heap_caps_register_failed_alloc_callback (esp_alloc_failed_hook_t callback)
registers a callback function to be invoked if a memory allocation operation fails.

参数 callback – caller defined callback to be invoked
返回 ESP_OK if callback was registered.

void *heap_caps_malloc (size_t size, uint32_t caps)
Allocate a chunk of memory which has the given capabilities.

Equivalent semantics to libc malloc(), for capability-aware memory.

参数
- size - Size, in bytes, of the amount of memory to allocate
- caps - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

返回 A pointer to the memory allocated on success, NULL on failure

void heap_caps_free (void *ptr)
Free memory previously allocated via heap_caps_malloc() or heap_caps_realloc().

Equivalent semantics to libc free(), for capability-aware memory.

In IDF, free(p) is equivalent to heap_caps_free(p).
### heap_caps_realloc

```c
void *heap_caps_realloc(void *ptr, size_t size, uint32_t caps)
```

Reallocate memory previously allocated via heap_caps_malloc() or heap_caps_realloc().

Equivalent semantics to libc realloc(), for capability-aware memory.

In IDF, realloc(p, s) is equivalent to heap_caps_realloc(p, s, MALLOC_CAP_8BIT).

- **ptr** - Pointer to previously allocated memory, or NULL for a new allocation.
- **size** - Size of the new buffer requested, or 0 to free the buffer.
- **caps** - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory desired for the new allocation.

### heap_caps_aligned_alloc

```c
void *heap_caps_aligned_alloc(size_t alignment, size_t size, uint32_t caps)
```

Allocate an aligned chunk of memory which has the given capabilities.

Equivalent semantics to libc aligned_alloc(), for capability-aware memory.

- **alignment** - How the pointer received needs to be aligned must be a power of two
- **size** - Size, in bytes, of the amount of memory to allocate
- **caps** - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

### heap_caps_aligned_free

```c
void heap_caps_aligned_free(void *ptr)
```

Used to deallocate memory previously allocated with heap_caps_aligned_alloc.

### heap_caps aligned calloc

```c
void *heap_caps_aligned_calloc(size_t alignment, size_t n, size_t size, uint32_t caps)
```

Allocate an aligned chunk of memory which has the given capabilities. The initialized value in the memory is set to zero.

- **alignment** - How the pointer received needs to be aligned must be a power of two
- **n** - Number of continuing chunks of memory to allocate
- **size** - Size, in bytes, of a chunk of memory to allocate
- **caps** - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

### heap_caps calloc

```c
void *heap_caps_calloc(size_t n, size_t size, uint32_t caps)
```

Allocate a chunk of memory which has the given capabilities. The initialized value in the memory is set to zero.

Equivalent semantics to libc calloc(), for capability-aware memory.

In IDF, calloc(p) is equivalent to heap_caps_calloc(p, MALLOC_CAP_8BIT).

- **n** - Number of continuing chunks of memory to allocate
Chapter 2. API

- **size**: Size, in bytes, of a chunk of memory to allocate
- **caps**: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

返回: A pointer to the memory allocated on success, NULL on failure

```c
size_t heap_caps_get_total_size(uint32_t caps)
```

Get the total size of all the regions that have the given capabilities.

This function takes all regions capable of having the given capabilities allocated in them and adds up the total space they have.

参数: `caps` - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
返回: total size in bytes

```c
size_t heap_caps_get_free_size(uint32_t caps)
```

Get the total free size of all the regions that have the given capabilities.

This function takes all regions capable of having the given capabilities allocated in them and adds up the free space they have.

备注: Note that because of heap fragmentation it is probably not possible to allocate a single block of memory of this size. Use heap_caps_get_largest_free_block() for this purpose.

```c
size_t heap_caps_get_minimum_free_size(uint32_t caps)
```

Get the total minimum free memory of all regions with the given capabilities.

This adds all the low watermarks of the regions capable of delivering the memory with the given capabilities.

备注: Note the result may be less than the global all-time minimum available heap of this kind, as “low watermarks” are tracked per-region. Individual regions’ heaps may have reached their “low watermarks” at different points in time. However, this result still gives a “worst case” indication for all-time minimum free heap.

```c
size_t heap_caps_get_largest_free_block(uint32_t caps)
```

Get the largest free block of memory able to be allocated with the given capabilities.

Returns the largest value of `s` for which `heap_caps_malloc(s, caps)` will succeed.

参数: `caps` - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
返回: Size of the largest free block in bytes.

```c
void heap_caps_get_info(multi_heap_info_t *info, uint32_t caps)
```

Get heap info for all regions with the given capabilities.

Calls multi_heap_info() on all heaps which share the given capabilities. The information returned is an aggregate across all matching heaps. The meanings of fields are the same as defined for `multi_heap_info_t`, except that `minimum_free_bytes` has the same caveats described in `heap_caps_get_minimum_free_size()`.

参数:
- `info` - Pointer to a structure which will be filled with relevant heap metadata.
- `caps` - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
void heap_caps_print_heap_info (uint32_t caps)
Print a summary of all memory with the given capabilities.
Calls multi_heap_info on all heaps which share the given capabilities, and prints a two-line summary for each, then a total summary.

参数 caps - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

bool heap_caps_check_integrity_all (bool print_errors)
Check integrity of all heap memory in the system.
Calls multi_heap_check on all heaps. Optionally print errors if heaps are corrupt.
Calling this function is equivalent to calling heap_caps_check_integrity with the caps argument set to MALLOC_CAP_INVALID.

备注: Please increase the value of CONFIG_ESP_INT_WDT_TIMEOUT_MS when using this API with PSRAM enabled.

参数 print_errors - Print specific errors if heap corruption is found.
返回 True if all heaps are valid, False if at least one heap is corrupt.

bool heap_caps_check_integrity (uint32_t caps, bool print_errors)
Check integrity of all heaps with the given capabilities.
Calls multi_heap_check on all heaps which share the given capabilities. Optionally print errors if the heaps are corrupt.
See also heap_caps_check_integrity_all to check all heap memory in the system and heap_caps_check_integrity_addr to check memory around a single address.

备注: Please increase the value of CONFIG_ESP_INT_WDT_TIMEOUT_MS when using this API with PSRAM capability flag.

参数
• caps - Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
• print_errors - Print specific errors if heap corruption is found.
返回 True if all heaps are valid, False if at least one heap is corrupt.

bool heap_caps_check_integrity_addr (intptr_t addr, bool print_errors)
Check integrity of heap memory around a given address.
This function can be used to check the integrity of a single region of heap memory, which contains the given address.
This can be useful if debugging heap integrity for corruption at a known address, as it has a lower overhead than checking all heap regions. Note that if the corrupt address moves around between runs (due to timing or other factors) then this approach won’t work, and you should call heap_caps_check_integrity or heap_caps_check_integrity_all instead.

备注: The entire heap region around the address is checked, not only the adjacent heap blocks.

参数
• addr - Address in memory. Check for corruption in region containing this address.
• print_errors - Print specific errors if heap corruption is found.
返回 True if the heap containing the specified address is valid, False if at least one heap is corrupt or the address doesn’t belong to a heap region.
void `heap_caps_malloc_extmem_enable` (size_t limit)

Enable malloc() in external memory and set limit below which malloc() attempts are placed in internal memory.

When external memory is in use, the allocation strategy is to initially try to satisfy smaller allocation requests with internal memory and larger requests with external memory. This sets the limit between the two, as well as generally enabling allocation in external memory.

参数 limit – Limit, in bytes.

void *`heap_caps_malloc_prefer` (size_t size, size_t num, ...)

Allocate a chunk of memory as preference in decreasing order.

Attention The variable parameters are bitwise OR of MALLOC_CAP_* flags indicating the type of memory. This API prefers to allocate memory with the first parameter. If failed, allocate memory with the next parameter. It will try in this order until allocating a chunk of memory successfully or fail to allocate memories with any of the parameters.

参数
- size – Size, in bytes, of the amount of memory to allocate
- num – Number of variable parameters

返回 A pointer to the memory allocated on success, NULL on failure

void *`heap_caps_realloc_prefer` (void *ptr, size_t size, size_t num, ...)

Reallocate a chunk of memory as preference in decreasing order.

参数
- ptr – Pointer to previously allocated memory, or NULL for a new allocation.
- size – Size of the new buffer requested, or 0 to free the buffer.
- num – Number of variable parameters

返回 Pointer to a new buffer of size ‘size’, or NULL if allocation failed.

void *`heap_caps_calloc_prefer` (size_t n, size_t size, size_t num, ...)

Allocate a chunk of memory as preference in decreasing order.

参数
- n – Number of continuing chunks of memory to allocate
- size – Size, in bytes, of a chunk of memory to allocate
- num – Number of variable parameters

返回 A pointer to the memory allocated on success, NULL on failure

void `heap_caps_dump` (uint32_t caps)

Dump the full structure of all heaps with matching capabilities.

Prints a large amount of output to serial (because of locking limitations, the output bypasses stdout/stderr). For each (variable sized) block in each matching heap, the following output is printed on a single line:

- Block address (the data buffer returned by malloc is 4 bytes after this if heap debugging is set to Basic, or 8 bytes otherwise).
- Data size (the data size may be larger than the size requested by malloc, either due to heap fragmentation or because of heap debugging level).
- Address of next block in the heap.
- If the block is free, the address of the next free block is also printed.

参数 caps – Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

void `heap_caps_dump_all` (void)

Dump the full structure of all heaps.

Covers all registered heaps. Prints a large amount of output to serial.
Output is the same as for heap_caps_dump.

```c
size_t heap_caps_get_allocated_size(void *ptr)
```

Return the size that a particular pointer was allocated with.

**备注:** The app will crash with an assertion failure if the pointer is not valid.

参数 `ptr` – Pointer to currently allocated heap memory. Must be a pointer value previously returned by heap_caps_malloc, malloc, calloc, etc. and not yet freed.

返回 Size of the memory allocated at this block.

**Macros**

**MALLOC_CAP_EXEC**

- Flags to indicate the capabilities of the various memory systems.
  - Memory must be able to run executable code

**MALLOC_CAP_32BIT**

- Memory must allow for aligned 32-bit data accesses.

**MALLOC_CAP_8BIT**

- Memory must allow for 8/16/...-bit data accesses.

**MALLOC_CAP_DMA**

- Memory must be able to accessed by DMA.

**MALLOC_CAP_PID2**

- Memory must be mapped to PID2 memory space (PIDs are not currently used)

**MALLOC_CAP_PID3**

- Memory must be mapped to PID3 memory space (PIDs are not currently used)

**MALLOC_CAP_PID4**

- Memory must be mapped to PID4 memory space (PIDs are not currently used)

**MALLOC_CAP_PID5**

- Memory must be mapped to PID5 memory space (PIDs are not currently used)

**MALLOC_CAP_PID6**

- Memory must be mapped to PID6 memory space (PIDs are not currently used)

**MALLOC_CAP_PID7**

- Memory must be mapped to PID7 memory space (PIDs are not currently used)

**MALLOC_CAP_SPIRAM**

- Memory must be in SPI RAM.

**MALLOC_CAP_INTERNAL**

- Memory must be internal; specifically it should not disappear when flash/spiram cache is switched off.
Chapter 2. API

**MALLOC_CAP_DEFAULT**
Memory can be returned in a non-capability-specific memory allocation (e.g. malloc(), calloc()) call.

**MALLOC_CAP_IRAM_8BIT**
Memory must be in IRAM and allow unaligned access.

**MALLOC_CAP_RETENTION**
Memory must be able to accessed by retention DMA.

**MALLOC_CAP_RTCRAM**
Memory must be in RTC fast memory.

**MALLOC_CAP_INVALID**
Memory can’t be used / list end marker.

**Type Definitions**

typedef void (*esp_alloc_failed_hook_t)(size_t size, uint32_t caps, const char *function_name)
callback called when an allocation operation fails, if registered

- **Param size** in bytes of failed allocation
- **Param caps** capabilities requested of failed allocation
- **Param function_name** function which generated the failure

**API Reference - Initialisation**

**Header File**

- components/heap/include/esp_heap_caps_init.h

**Functions**

void **heap_caps_init** (void)
Initialize the capability-aware heap allocator.

This is called once in the IDF startup code. Do not call it at other times.

void **heap_caps_enable_nonos_stack_heaps** (void)
Enable heap(s) in memory regions where the startup stacks are located.

On startup, the pro/app CPUs have a certain memory region they use as stack, so we cannot do allocations in the regions these stack frames are. When FreeRTOS is completely started, they do not use that memory anymore and heap(s) there can be enabled.

**esp_err_t** **heap_caps_add_region** (intptr_t start, intptr_t end)
Add a region of memory to the collection of heaps at runtime.

Most memory regions are defined in soc_memory_layout.c for the SoC, and are registered via heap_caps_init(). Some regions can’t be used immediately and are later enabled via heap_caps_enable_nonos_stack_heaps().

Call this function to add a region of memory to the heap at some later time.

This function does not consider any of the “reserved” regions or other data in soc_memory_layout, caller needs to consider this themselves.

All memory within the region specified by start & end parameters must be otherwise unused.

The capabilities of the newly registered memory will be determined by the start address, as looked up in the regions specified in soc_memory_layout.c.
Use `heap_caps_add_region_with_caps()` to register a region with custom capabilities.

### Parameters

- **start** – Start address of new region.
- **end** – End address of new region.

### Returns

- **ESP_OK** on success
- **ESP_ERR_INVALID_ARG** if a parameter is invalid
- **ESP_ERR_NOT_FOUND** if the specified start address doesn’t reside in a known region, or any error returned by `heap_caps_add_region_with_caps()`.

**`esp_err_t heap_caps_add_region_with_caps(const uint32_t caps[], intptr_t start, intptr_t end)`**

Add a region of memory to the collection of heaps at runtime, with custom capabilities.

Similar to `heap_caps_add_region()`, only custom memory capabilities are specified by the caller.

### Example

<table>
<thead>
<tr>
<th>Existing region: 0x1000 &lt;-&gt; 0x3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>New region: 0x1000 &lt;-&gt; 0x3000 (Allowed)</td>
</tr>
<tr>
<td>New region: 0x1000 &lt;-&gt; 0x2000 (Allowed)</td>
</tr>
<tr>
<td>New region: 0x0000 &lt;-&gt; 0x1000 (Allowed)</td>
</tr>
<tr>
<td>New region: 0x3000 &lt;-&gt; 0x4000 (Allowed)</td>
</tr>
<tr>
<td>New region: 0x0000 &lt;-&gt; 0x2000 (NOT Allowed)</td>
</tr>
<tr>
<td>New region: 0x0000 &lt;-&gt; 0x4000 (NOT Allowed)</td>
</tr>
<tr>
<td>New region: 0x1000 &lt;-&gt; 0x4000 (NOT Allowed)</td>
</tr>
<tr>
<td>New region: 0x2000 &lt;-&gt; 0x4000 (NOT Allowed)</td>
</tr>
</tbody>
</table>
Chapter 2. API 参考

API Reference - Multi Heap API

(Note: The multi heap API is used internally by the heap capabilities allocator. Most IDF programs will never need to call this API directly.)

Header File

- components/heap/include/multi_heap.h

Functions

void *multi_heap_aligned_alloc(multi_heap_handle_t heap, size_t size, size_t alignment)
allocate a chunk of memory with specific alignment

参数
- **heap** - Handle to a registered heap.
- **size** - size in bytes of memory chunk
- **alignment** - how the memory must be aligned

返回 pointer to the memory allocated, NULL on failure

void *multi_heap_malloc(multi_heap_handle_t heap, size_t size)
malloc() a buffer in a given heap

Semantics are the same as standard malloc(), only the returned buffer will be allocated in the specified heap.

参数
- **heap** - Handle to a registered heap.
- **size** - Size of desired buffer.

返回 Pointer to new memory, or NULL if allocation fails.

void multi_heap_aligned_free(multi_heap_handle_t heap, void *p)
free() a buffer aligned in a given heap.

备注：This function is deprecated, consider using multi_heap_free() instead

参数
- **heap** - Handle to a registered heap.
- **p** - NULL, or a pointer previously returned from multi_heap_aligned_alloc() for the same heap.

void multi_heap_free(multi_heap_handle_t heap, void *p)
free() a buffer in a given heap.

Semantics are the same as standard free(), only the argument `p` must be NULL or have been allocated in the specified heap.

参数
- **heap** - Handle to a registered heap.
- **p** - NULL, or a pointer previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.

void *multi_heap_realloc(multi_heap_handle_t heap, void *p, size_t size)
realloc() a buffer in a given heap.

Semantics are the same as standard realloc(), only the argument `p` must be NULL or have been allocated in the specified heap.

参数
- **heap** - Handle to a registered heap.
- **p** - NULL, or a pointer previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.
Chapter 2. API Reference

- **size** - Desired new size for buffer.

Returns: New buffer of ‘size’ containing contents of ‘p’, or NULL if reallocation failed.

```c
size_t multi_heap_get_allocated_size(multi_heap_handle_t heap, void *p)
```

Return the size that a particular pointer was allocated with.

Parameters:
- `heap` - Handle to a registered heap.
- `p` - Pointer, must have been previously returned from `multi_heap_malloc()` or `multi_heap_realloc()` for the same heap.

Returns: Size of the memory allocated at this block. May be more than the original size argument, due to padding and minimum block sizes.

```c
multi_heap_handle_t multi_heap_register(void *start, size_t size)
```

Register a new heap for use.

This function initialises a heap at the specified address, and returns a handle for future heap operations.

There is no equivalent function for deregistering a heap - if all blocks in the heap are free, you can immediately start using the memory for other purposes.

Parameters:
- `start` - Start address of the memory to use for a new heap.
- `size` - Size (in bytes) of the new heap.

Returns: Handle of a new heap ready for use, or NULL if the heap region was too small to be initialised.

```c
void multi_heap_set_lock(multi_heap_handle_t heap, void *lock)
```

Associate a private lock pointer with a heap.

The lock argument is supplied to the `MULTI_HEAP_LOCK()` and `MULTI_HEAP_UNLOCK()` macros, defined in `multi_heap_platform.h`.

The lock in question must be recursive.

When the heap is first registered, the associated lock is NULL.

Parameters:
- `heap` - Handle to a registered heap.
- `lock` - Optional pointer to a locking structure to associate with this heap.

```c
void multi_heap_dump(multi_heap_handle_t heap)
```

Dump heap information to stdout.

For debugging purposes, this function dumps information about every block in the heap to stdout.

Parameters: `heap` - Handle to a registered heap.

```c
bool multi_heap_check(multi_heap_handle_t heap, bool print_errors)
```

Check heap integrity.

Walks the heap and checks all heap data structures are valid. If any errors are detected, an error-specific message can be optionally printed to stderr. Print behaviour can be overridden at compile time by defining `MULTI_CHECK_FAILPRINTF` in `multi_heap_platform.h`.

**Remark:** This function is not thread-safe as it sets a global variable with the value of `print_errors`.

Parameters:
- `heap` - Handle to a registered heap.
- `print_errors` - If true, errors will be printed to stderr.

Returns: true if heap is valid, false otherwise.
Chapter 2. API

size_t multi_heap_free_size (multi_heap_handle_t heap)

    Return free heap size.
    
    Returns the number of bytes available in the heap.
    
    Equivalent to the total_free_bytes member returned by multi_heap_get_heap_info().
    
    Note that the heap may be fragmented, so the actual maximum size for a single malloc() may be lower. To know this size, see the largest_free_block member returned by multi_heap_get_heap_info().

    参数 heap - Handle to a registered heap.
    返回 Number of free bytes.

size_t multi_heap_minimum_free_size (multi_heap_handle_t heap)

    Return the lifetime minimum free heap size.
    
    Equivalent to the minimum_free_bytes member returned by multi_heap_get_info().
    
    Returns the lifetime “low watermark” of possible values returned from multi_free_heap_size(), for the specified heap.

    参数 heap - Handle to a registered heap.
    返回 Number of free bytes.

void multi_heap_get_info (multi_heap_handle_t heap, multi_heap_info_t *info)

    Return metadata about a given heap.
    
    Fills a multi_heap_info_t structure with information about the specified heap.

    参数
    • heap - Handle to a registered heap.
    • info - Pointer to a structure to fill with heap metadata.

Structures

struct multi_heap_info_t

    Structure to access heap metadata via multi_heap_get_info.

Public Members

size_t total_free_bytes

    Total free bytes in the heap. Equivalent to multi_free_heap_size().

size_t total_allocated_bytes

    Total bytes allocated to data in the heap.

size_t largest_free_block

    Size of the largest free block in the heap. This is the largest malloc-able size.

size_t minimum_free_bytes

    Lifetime minimum free heap size. Equivalent to multi_minimum_free_heap_size().

size_t allocated_blocks

    Number of (variable size) blocks allocated in the heap.

size_t free_blocks

    Number of (variable size) free blocks in the heap.
The total number of (variable size) blocks in the heap.

**Type Definitions**

typedef struct multi_heap_info *multi_heap_handle_t

Opaque handle to a registered heap.

### 2.10.13 Heap Memory Debugging

**Overview**

ESP-IDF integrates tools for requesting heap information, detecting heap corruption, and tracing memory leaks. These can help track down memory-related bugs.

For general information about the heap memory allocator, see the *Heap Memory Allocation* page.

**Heap Information**

To obtain information about the state of the heap:

- `xPortGetFreeHeapSize()` is a FreeRTOS function which returns the number of free bytes in the (data memory) heap. This is equivalent to calling `heap_caps_get_free_size(MALLOC_CAP_8BIT)`.
- `heap_caps_get_free_size()` can also be used to return the current free memory for different memory capabilities.
- `heap_caps_get_largest_free_block()` can be used to return the largest free block in the heap. This is the largest single allocation which is currently possible. Tracking this value and comparing to total free heap allows you to detect heap fragmentation.
- `xPortGetMinimumEverFreeHeapSize()` and the related `heap_caps_get_minimum_free_size()` can be used to track the heap “low watermark” since boot.
- `heap_caps_get_info()` returns a `multi_heap_info_t` structure which contains the information from the above functions, plus some additional heap-specific data (number of allocations, etc.).
- `heap_caps_print_heap_info()` prints a summary to stdout of the information returned by `heap_caps_get_info()`.
- `heap_caps_dump()` and `heap_caps_dump_all()` will output detailed information about the structure of each block in the heap. Note that this can be large amount of output.

**Heap Corruption Detection**

Heap corruption detection allows you to detect various types of heap memory errors:

- Out of bounds writes & buffer overflow.
- Writes to freed memory.
- Reads from freed or uninitialized memory.

**Assertions** The heap implementation (multi_heap.c, etc.) includes a lot of assertions which will fail if the heap memory is corrupted. To detect heap corruption most effectively, ensure that assertions are enabled in the project configuration menu under Compiler options -> `CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL`.

If a heap integrity assertion fails, a line will be printed like `CORRUPT HEAP: multi_heap.c:225 detected at 0x3ffbb71c`. The memory address which is printed is the address of the heap structure which has corrupt content.

It’s also possible to manually check heap integrity by calling `heap_caps_check_integrity_all()` or related functions. This function checks all of requested heap memory for integrity, and can be used even if assertions are disabled. If the integrity check prints an error, it will also contain the address(es) of corrupt heap structures.
Memory Allocation Failed Hook  Users can use `heap_caps_register_failed_alloc_callback()` to register a callback that will be invoked every time an allocation operation fails.

Additionally, users can enable the generation of a system abort if an allocation operation fails by following the steps below: - In the project configuration menu, navigate to Component config -> Heap Memory Debugging and select Abort if memory allocation fails option (see `CONFIG_HEAP_ABORT_WHEN_ALLOCATION_FAILS`).

The example below shows how to register an allocation failure callback:

```c
#include "esp_heap_caps.h"

void heap_caps_alloc_failed_hook(size_t requested_size, uint32_t caps, const char *function_name)
{  
  printf("%s was called but failed to allocate %d bytes with 0x%X capabilities. \n", function_name, requested_size, caps);
}

void app_main()
{
  ...
  esp_err_t error = heap_caps_register_failed_alloc_callback(heap_caps_alloc_failed_hook);
  ...
  void *ptr = heap_caps_malloc(allocation_size, MALLOC_CAP_DEFAULT);
  ...
}
```

Finding Heap Corruption  Memory corruption can be one of the hardest classes of bugs to find and fix, as one area of memory can be corrupted from a totally different place. Some tips:

- A crash with a CORRUPT HEAP: message will usually include a stack trace, but this stack trace is rarely useful. The crash is the symptom of memory corruption when the system realises the heap is corrupt, but usually the corruption happened elsewhere and earlier in time.
- Increasing the Heap memory debugging Configuration level to “Light impact” or “Comprehensive” can give you a more accurate message with the first corrupt memory address.
- Adding regular calls to `heap_caps_check_integrity_all()` or `heap_caps_check_integrity_addr()` in your code will help you pin down the exact time that the corruption happened. You can move these checks around to “close in on” the section of code that corrupted the heap.
- Based on the memory address which is being corrupted, you can use JTAG debugging to set a watchpoint on this address and have the CPU halt when it is written to.
- If you don’t have JTAG, but you do know roughly when the corruption happens, then you can set a watchpoint in software just beforehand via `esp_cpu_set_watchpoint()`. A fatal exception will occur when the watchpoint triggers. The following is an example of how to use the function - `esp_cpu_set_watchpoint(0, (void *)addr, 4, ESP_WATCHPOINT_STORE)`. Note that watchpoints are per-CPU and are set on the current running CPU only, so if you don’t know which CPU is corrupting memory then you will need to call this function on both CPUs.
- For buffer overflows, heap tracing in HEAP_TRACE_ALL mode lets you see which callers are allocating which addresses from the heap. See Heap Tracing To Find Heap Corruption for more details. If you can find the function which allocates memory with an address immediately before the address which is corrupted, this will probably be the function which overflows the buffer.
- Calling `heap_caps_dump()` or `heap_caps_dump_all()` can give an indication of what heap blocks are surrounding the corrupted region and may have overflowed/underflowed/etc.

Configuration  Temporarily increasing the heap corruption detection level can give more detailed information about heap corruption errors.
In the project configuration menu, under Component config there is a menu Heap memory debugging. The setting CONFIG_HEAP_CORRUPTION_DETECTION can be set to one of three levels:

**Basic (no poisoning)** This is the default level. No special heap corruption features are enabled, but provided assertions are enabled (the default configuration) then a heap corruption error will be printed if any of the heap’s internal data structures appear overwritten or corrupted. This usually indicates a buffer overrun or out of bounds write.

If assertions are enabled, an assertion will also trigger if a double-free occurs (the same memory is freed twice). Calling heap_caps_check_integrity() in Basic mode will check the integrity of all heap structures, and print errors if any appear to be corrupted.

**Light Impact** At this level, heap memory is additionally “poisoned” with head and tail “canary bytes” before and after each block which is allocated. If an application writes outside the bounds of allocated buffers, the canary bytes will be corrupted and the integrity check will fail.

The head canary word is 0xABBA1234 (3412BAAB in byte order), and the tail canary word is 0xBAAD5678 (7856ADBA in byte order).

“Basic” heap corruption checks can also detect most out of bounds writes, but this setting is more precise as even a single byte overrun can be detected. With Basic heap checks, the number of overrun bytes before a failure is detected will depend on the properties of the heap.

Enabling “Light Impact” checking increases memory usage, each individual allocation will use 9 to 12 additional bytes of memory (depending on alignment).

Each time free() is called in Light Impact mode, the head and tail canary bytes of the buffer being freed are checked against the expected values.

When heap_caps_check_integrity() is called, all allocated blocks of heap memory have their canary bytes checked against the expected values.

In both cases, the check is that the first 4 bytes of an allocated block (before the buffer returned to the user) should be the word 0xABBA1234. Then the last 4 bytes of the allocated block (after the buffer returned to the user) should be the word 0xBAAD5678.

Different values usually indicate buffer underrun or overrun, respectively.

**Comprehensive** This level incorporates the “light impact” detection features plus additional checks for uninitialised-access and use-after-free bugs. In this mode, all freshly allocated memory is filled with the pattern 0xCE, and all freed memory is filled with the pattern 0xFE.

Enabling “Comprehensive” detection has a substantial runtime performance impact (as all memory needs to be set to the allocation patterns each time a malloc/free completes, and the memory also needs to be checked each time.) However, it allows easier detection of memory corruption bugs which are much more subtle to find otherwise. It is recommended to only enable this mode when debugging, not in production.

**Crashes in Comprehensive Mode** If an application crashes reading/writing an address related to 0xCECECECECE in Comprehensive mode, this indicates it has read uninitialized memory. The application should be changed to either use calloc() (which zeroes memory), or initialize the memory before using it. The value 0xCECECECECE may also be seen in stack-allocated automatic variables, because in IDF most task stacks are originally allocated from the heap and in C stack memory is uninitialized by default.

If an application crashes and the exception register dump indicates that some addresses or values were 0xFEFEFEFEFE, this indicates it is reading heap memory after it has been freed (a “use after free bug”. The application should be changed to not access heap memory after it has been freed.

If a call to malloc() or realloc() causes a crash because it expected to find the pattern 0xFEFEFEFEFE in free memory and a different pattern was found, then this indicates the app has a use-after-free bug where it is writing to memory which has already been freed.
Manual Heap Checks in Comprehensive Mode  Calls to `heap_caps_check_integrity()` may print errors relating to 0xFEFEFEFE, 0xABBA1234 or 0xBAAD5678. In each case the checker is expecting to find a given pattern, and will error out if this is not found:

- For free heap blocks, the checker expects to find all bytes set to 0xFE. Any other values indicate a use-after-free bug where free memory has been incorrectly overwritten.
- For allocated heap blocks, the behaviour is the same as for Light Impact mode. The canary bytes 0xABBA1234 and 0xBAAD5678 are checked at the head and tail of each allocated buffer, and any variation indicates a buffer overrun/underrun.

Heap Task Tracking

Heap Task Tracking can be used to get per task info for heap memory allocation. Application has to specify the heap capabilities for which the heap allocation is to be tracked.

Example code is provided in `system/heap_task_tracking`

Heap Tracing

Heap Tracing allows tracing of code which allocates/frees memory. Two tracing modes are supported:

- Standalone. In this mode trace data are kept on-board, so the size of gathered information is limited by the buffer assigned for that purposes. Analysis is done by the on-board code. There are a couple of APIs available for accessing and dumping collected info.
- Host-based. This mode does not have the limitation of the standalone mode, because trace data are sent to the host over JTAG connection using app_trace library. Later on they can be analysed using special tools.

Heap tracing can perform two functions:

- Leak checking: find memory which is allocated and never freed.
- Heap use analysis: show all functions that are allocating/freeing memory while the trace is running.

How To Diagnose Memory Leaks  If you suspect a memory leak, the first step is to figure out which part of the program is leaking memory. Use the `xPortGetFreeHeapSize()`, `heap_caps_get_free_size()`, or related functions to track memory use over the life of the application. Try to narrow the leak down to a single function or sequence of functions where free memory always decreases and never recovers.

Standalone Mode  Once you’ve identified the code which you think is leaking:

- In the project configuration menu, navigate to Component settings -> Heap Memory Debugging -> Heap tracing and select Standalone option (see `CONFIG_HEAP_TRACING_DEST`).
- Call the function `heap_trace_init_standalone()` early in the program, to register a buffer which can be used to record the memory trace.
- Call the function `heap_trace_start()` to begin recording all mallocs/frees in the system. Call this immediately before the piece of code which you suspect is leaking memory.
- Call the function `heap_trace_stop()` to stop the trace once the suspect piece of code has finished executing.
- Call the function `heap_trace_dump()` to dump the results of the heap trace.

An example:

```c
#include "esp_heap_trace.h"

#define NUM_RECORDS 100
static heap_trace_record_t trace_record[NUM_RECORDS]; // This buffer must be in internal RAM
...
```

(下页续接)
Chapter 2. API

gॉ

(线索)

```c
void app_main()
{
    ...  
    ESP_ERROR_CHECK( heap_trace_init_standalone(trace_record, NUM_RECORDS) );
    ...
}

void some_function()
{
    ESP_ERROR_CHECK( heap_trace_start(HEAP_TRACE_LEAKS) );
    do_something_you_suspect_is_leaking();
    ESP_ERROR_CHECK( heap_trace_stop() );
    heap_trace_dump();
    ...
}
```

The output from the heap trace will look something like this:

```
2 allocations trace (100 entry buffer)
32 bytes (@ 0x3ffaf214) allocated CPU 0 ccoun 0x2e9b7384 caller...
    0x400d276d: leak_some_memory at /path/to/idf/examples/get-started/blink/main/.
    ~blink.c:27
0x400d27c1: blink_task at /path/to/idf/examples/get-started/blink/main/./blink.c:52
8 bytes (@ 0x3ffaf804) allocated CPU 0 ccoun 0x2e9b79c0 caller...
    0x400d2776: leak_some_memory at /path/to/idf/examples/get-started/blink/main/.
    ~blink.c:29
0x400d27c1: blink_task at /path/to/idf/examples/get-started/blink/main/./blink.c:52
40 bytes ‘leaked’ in trace (2 allocations)
total allocations 2 total frees 0
```

(Above example output is using IDF Monitor to automatically decode PC addresses to their source files & line number.)

The first line indicates how many allocation entries are in the buffer, compared to its total size.

In HEAP_TRACE_LEAKS mode, for each traced memory allocation which has not already been freed a line is printed with:

- XX bytes is the number of bytes allocated
- @ 0x... is the heap address returned from malloc/calloc.
- CPU x is the CPU (0 or 1) running when the allocation was made.
- ccoun 0x... is the CCOUNT (CPU cycle count) register value when the allocation was made. Is different for CPU 0 vs CPU 1.
- caller 0x... gives the call stack of the call to malloc()/free(), as a list of PC addresses. These can be decoded to source files and line numbers, as shown above.

The depth of the call stack recorded for each trace entry can be configured in the project configuration menu, under Heap Memory Debugging -> Enable heap tracing -> Heap tracing stack depth. Up to 10 stack frames can be recorded for each allocation (the default is 2). Each additional stack frame increases the memory usage of each heap_trace_record_t record by eight bytes.

Finally, the total number of ‘leaked’ bytes (bytes allocated but not freed while trace was running) is printed, and the total number of allocations this represents.
A warning will be printed if the trace buffer was not large enough to hold all the allocations which happened. If you see this warning, consider either shortening the tracing period or increasing the number of records in the trace buffer.

### Host-Based Mode

Once you’ve identified the code which you think is leaking:

- In the project configuration menu, navigate to **Component settings** -> **Heap Memory Debugging** -> **CONFIG_HEAP_TRACING_DEST** and select **Host-Based**.
- In the project configuration menu, navigate to **Component settings** -> **Application Level Tracing** -> **CONFIG_APPTRACE_DESTINATION** and select **Trace memory**.
- In the project configuration menu, navigate to **Component settings** -> **Application Level Tracing** -> **FreeRTOS SystemView Tracing** and enable **CONFIG_APPTRACE_SV_ENABLE**.
- Call the function `heap_trace_init_tohost()` early in the program, to initialize JTAG heap tracing module.
- Call the function `heap_trace_start()` to begin recording all mallocs/frees in the system. Call this immediately before the piece of code which you suspect is leaking memory. In host-based mode, the argument to this function is ignored, and the heap tracing module behaves like **HEAP_TRACE_ALL** was passed: all allocations and deallocations are sent to the host.
- Call the function `heap_trace_stop()` to stop the trace once the suspect piece of code has finished executing.

An example:

```c
#include "esp_heap_trace.h"
...
void app_main()
{
    ...
    ESP_ERROR_CHECK( heap_trace_init_tohost() );
    ...
}
void some_function()
{
    ESP_ERROR_CHECK( heap_trace_start(HEAP_TRACE_LEAKS) );
    do_something_you_suspect_is_leaking();
    ESP_ERROR_CHECK( heap_trace_stop() );
    ...
}
```

To gather and analyse heap trace do the following on the host:

1. Build the program and download it to the target as described in *Getting Started Guide*.
2. Run OpenOCD (see *JTAG Debugging*).

**备注:** In order to use this feature you need OpenOCD version v0.10.0-esp32-20181105 or later.

3. You can use GDB to start and/or stop tracing automatically. To do this you need to prepare special `gdbinit` file:

```bash
target remote :3333
mon reset halt
flushregs
tb heap_trace_start
commands
```

(下页继续)
Using this file GDB will connect to the target, reset it, and start tracing when program hits breakpoint at `heap_trace_start()`. Tracedata will be saved to `/tmp/heap_log.svdat`. Tracing will be stopped when program hits breakpoint at `heap_trace_stop()`.

4. Run GDB using the following command:
   ```
   xtensa-esp32-elf-gdb -x gdbinit <path/to/program/elf>
   ```
5. Quit GDB when program stops at `heap_trace_stop()`. Trace data are saved in `/tmp/heap.svdat`
6. Run processing script:
   ```
   $IDF_PATH/tools/esp_app_trace/sysviewtrace_proc.py -p -b <path/to/program/elf> /tmp/heap_log.svdat
   ```

The output from the heap trace will look something like this:

```
Parse trace from '/tmp/heap.svdat'...
Stop parsing trace. (Timeout 0.000000 sec while reading 1 bytes!)
Process events from '[/tmp/heap.svdat]'...
[0.002244575] HEAP: Allocated 1 bytes @ 0x3ffaffd8 from task "alloc" on core 0 by:
/home/user/projects/esp-idf/examples/system/sysview_tracing_heap_log/main/
→sysview_heap_log.c:47
/home/user/projects/esp-idf/components/freertos/port.c:355 (discriminator 1)
[0.002258425] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp-idf/examples/system/sysview_tracing_heap_log/main/
→sysview_heap_log.c:48
/home/user/projects/esp-idf/components/freertos/port.c:355 (discriminator 1)
[0.002563725] HEAP: Freed bytes @ 0x3ffaffe0 from task "free" on core 0 by:
/home/user/projects/esp-idf/examples/system/sysview_tracing_heap_log/main/
→sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp-idf/components/freertos/port.c:355 (discriminator 1)
[0.002782950] HEAP: Freed bytes @ 0x3ffb40b8 from task "main" on core 0 by:
/home/user/projects/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp-idf/components/freertos/tasks.c:4590
[0.002798700] HEAP: Freed bytes @ 0x3ffb50bc from task "main" on core 0 by:
/home/user/projects/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp-idf/components/freertos/tasks.c:4590
[0.102436025] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp-idf/examples/system/sysview_tracing_heap_log/main/
→sysview_heap_log.c:47
/home/user/projects/esp-idf/components/freertos/port.c:355 (discriminator 1)
[0.102449800] HEAP: Allocated 4 bytes @ 0x3ffaffe8 from task "alloc" on core 0 by:
/home/user/projects/esp-idf/examples/system/sysview_tracing_heap_log/main/
→sysview_heap_log.c:48
/home/user/projects/esp-idf/components/freertos/port.c:355 (discriminator 1)
[0.102666150] HEAP: Freed bytes @ 0x3ffaffe8 from task "free" on core 0 by:
/home/user/projects/esp-idf/examples/system/sysview_tracing_heap_log/main/
→sysview_heap_log.c:31 (discriminator 9)
```
Heap Tracing To Find Heap Corruption

Heap tracing can also be used to help track down heap corruption. When a region in heap is corrupted, it may be from some other part of the program which allocated memory at a
nearby address.
If you have some idea at what time the corruption occurred, enabling heap tracing in \texttt{HEAP\_TRACE\_ALL} mode allows you to record all the functions which allocated memory, and the addresses of the allocations.

Using heap tracing in this way is very similar to memory leak detection as described above. For memory which is allocated and not freed, the output is the same. However, records will also be shown for memory which has been freed.

\textbf{Performance Impact}  
Enabling heap tracing in menuconfig increases the code size of your program, and has a very small negative impact on performance of heap allocation/free operations even when heap tracing is not running.

When heap tracing is running, heap allocation/free operations are substantially slower than when heap tracing is stopped. Increasing the depth of stack frames recorded for each allocation (see above) will also increase this performance impact.

\textbf{False-Positive Memory Leaks}  
Not everything printed by \texttt{heap\_trace\_dump()} is necessarily a memory leak. Among things which may show up here, but are not memory leaks:

- Any memory which is allocated after \texttt{heap\_trace\_start()} but then freed after \texttt{heap\_trace\_stop()} will appear in the leak dump.
- Allocations may be made by other tasks in the system. Depending on the timing of these tasks, it’s quite possible this memory is freed after \texttt{heap\_trace\_stop()} is called.
- The first time a task uses stdout - for example, when it calls \texttt{printf()} - a lock (RTOS mutex semaphore) is allocated by the libc. This allocation lasts until the task is deleted.
- Certain uses of \texttt{printf()}, such as printing floating point numbers, will allocate some memory from the heap on demand. These allocations last until the task is deleted.
- The Bluetooth, Wi-Fi, and TCP/IP libraries will allocate heap memory buffers to handle incoming or outgoing data. These memory buffers are usually short-lived, but some may be shown in the heap leak trace if the data was received/transmitted by the lower levels of the network while the leak trace was running.
- TCP connections will continue to use some memory after they are closed, because of the \texttt{TIME\_WAIT} state. After the \texttt{TIME\_WAIT} period has completed, this memory will be freed.

One way to differentiate between “real” and “false positive” memory leaks is to call the suspect code multiple times while tracing is running, and look for patterns (multiple matching allocations) in the heap trace output.

\textbf{API Reference - Heap Tracing}

\textbf{Header File}
- \texttt{components/heap/include/esp\_heap\_trace.h}

\textbf{Functions}
\begin{verbatim}
esp_err_t heap_trace_init_standalone (heap_trace_record_t *record_buffer, size_t num_records)
Initialise heap tracing in standalone mode.

This function must be called before any other heap tracing functions.

To disable heap tracing and allow the buffer to be freed, stop tracing and then call
heap_trace_init_standalone(NULL, 0);
\end{verbatim}

\begin{itemize}
\item \texttt{record_buffer} – Provide a buffer to use for heap trace data. Must remain valid any time heap tracing is enabled, meaning it must be allocated from internal memory not in PSRAM.
\item \texttt{num_records} – Size of the heap trace buffer, as number of record structures.
\end{itemize}

\begin{itemize}
\item ESP\_ERR\_NOT\_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
\end{itemize}
Chapter 2. API 参考

• ESP_ERR_INVALID_STATE Heap tracing is currently in progress.
• ESP_OK Heap tracing initialised successfully.

\texttt{esp_err_t heap_trace_init_tohost (void)}

Initialise heap tracing in host-based mode.

- ESP_ERR_INVALID_STATE Heap tracing is currently in progress.
- ESP_OK Heap tracing initialised successfully.

\texttt{esp_err_t heap_trace_start (heap_trace_mode_t mode)}

Start heap tracing. All heap allocations & frees will be traced, until \texttt{heap_trace_stop()} is called.

\textbf{备注:} heap_trace_init_standalone() must be called to provide a valid buffer, before this function is called.

\textbf{备注:} Calling this function while heap tracing is running will reset the heap trace state and continue tracing.

参数 \texttt{mode} - Mode for tracing.
- HEAP_TRACE_ALL means all heap allocations and frees are traced.
- HEAP_TRACE_LEAKS means only suspected memory leaks are traced. (When memory is freed, the record is removed from the trace buffer.)

\textbf{备注:} ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE A non-zero-length buffer has not been set via heap_trace_init_standalone().
- ESP_OK Tracing is started.

\texttt{esp_err_t heap_trace_stop (void)}

Stop heap tracing.

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was not in progress.
- ESP_OK Heap tracing stopped.

\texttt{esp_err_t heap_trace_resume (void)}

Resume heap tracing which was previously stopped.

Unlike \texttt{heap_trace_start()}, this function does not clear the buffer of any pre-existing trace records.

The heap trace mode is the same as when \texttt{heap_trace_start()} was last called (or HEAP_TRACE_ALL if \texttt{heap_trace_start()} was never called).

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was already started.
- ESP_OK Heap tracing resumed.

\texttt{size_t heap_trace_get_count (void)}

Return number of records in the heap trace buffer.

It is safe to call this function while heap tracing is running.
**esp_err_t heap_trace_get** (size_t index, heap_trace_record_t *record)

Return a raw record from the heap trace buffer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Index (zero-based) of the record to return.</td>
</tr>
<tr>
<td>record [out]</td>
<td>Record where the heap trace record will be copied.</td>
</tr>
</tbody>
</table>

备注：It is safe to call this function while heap tracing is running, however in HEAP_TRACE_LEAK mode record indexing may skip entries unless heap tracing is stopped first.

-exit-

**void heap_trace_dump** (void)

Dump heap trace record data to stdout.

备注：It is safe to call this function while heap tracing is running, however in HEAP_TRACE_LEAK mode the dump may skip entries unless heap tracing is stopped first.

**Structures**

```c
struct heap_trace_record_t
```

Trace record data type. Stores information about an allocated region of memory.

**Public Members**

```c
uint32_t ccount
```

C_COUNT of the CPU when the allocation was made. LSB (bit value 1) is the CPU number (0 or 1).

```c
void *address
```

Address which was allocated.

```c
size_t size
```

Size of the allocation.

```c
void *allocated_by[CONFIG_HEAP_TRACING_STACK_DEPTH]
```

Call stack of the caller which allocated the memory.

```c
void *freed_by[CONFIG_HEAP_TRACING_STACK_DEPTH]
```

Call stack of the caller which freed the memory (all zero if not freed.)

**Macros**

```c
CONFIG_HEAP_TRACING_STACK_DEPTH
```
Chapter 2. API

Enumerations

defined

\textbf{heap\_trace\_mode\_t}

Values:

enumerator \textbf{HEAP\_TRACE\_ALL}

enumerator \textbf{HEAP\_TRACE\_LEAKS}

2.10.14 High Resolution Timer (ESP Timer)

Overview

Although FreeRTOS provides software timers, these timers have a few limitations:

- Maximum resolution is equal to RTOS tick period
- Timer callbacks are dispatched from a low-priority task

Hardware timers are free from both of the limitations, but often they are less convenient to use. For example, application components may need timer events to fire at certain times in the future, but the hardware timer only contains one “compare” value used for interrupt generation. This means that some facility needs to be built on top of the hardware timer to manage the list of pending events and dispatch the callbacks for these events as corresponding hardware interrupts happen.

An interrupt level of the handler depends on the \textbf{CONFIG\_ESP\_TIMER\_INTERUPT\_LEVEL} option. It allows to set this: 1, 2 or 3 level (by default 1). Raising the level, the interrupt handler can reduce the timer processing delay.

\textbf{esp\_timer} set of APIs provides one-shot and periodic timers, microsecond time resolution, and 64-bit range.

Internally, \textbf{esp\_timer} uses a 64-bit hardware timer, where the implementation depends on the target. LAC timer is used for ESP32.

Timer callbacks can be dispatched by two methods:

- \textbf{ESP\_TIMER\_TASK}. Timers are dispatched from a high-priority \textbf{esp\_timer} task. Because all the callbacks are dispatched from the same task, it is recommended to only do the minimal possible amount of work from the callback itself, posting an event to a lower priority task using a queue instead.

- \textbf{ESP\_TIMER\_ISR}. Available only if \textbf{CONFIG\_ESP\_TIMER\_SUPPORTS\_ISR\_DISPATCH\_METHOD} is enabled (by default disabled).

\textbf{ESP\_TIMER\_TASK}. Timer callbacks are dispatched from a high-priority \textbf{esp\_timer} task. Because all the callbacks are dispatched from the same task, it is recommended to only do the minimal possible amount of work from the callback itself, posting an event to a lower priority task using a queue instead.

If other tasks with priority higher than \textbf{esp\_timer} are running, callback dispatching will be delayed until \textbf{esp\_timer} task has a chance to run. For example, this will happen if an SPI Flash operation is in progress.

\textbf{ESP\_TIMER\_ISR}. Timer callbacks are dispatched directly from the timer interrupt handler. This method is useful for some simple callbacks which aim for lower latency.

Creating and starting a timer, and dispatching the callback takes some time. Therefore, there is a lower limit to the timeout value of one-shot \textbf{esp\_timer}. If \textbf{esp\_timer\_start\_once()} is called with a timeout value less than 20us, the callback will be dispatched only after approximately 20us.

Periodic \textbf{esp\_timer} also imposes a 50us restriction on the minimal timer period. Periodic software timers with period of less than 50us are not practical since they would consume most of the CPU time. Consider using dedicated hardware peripherals or DMA features if you find that a timer with small period is required.

Using \textbf{esp\_timer} APIs

Single timer is represented by \textbf{esp\_timer\_handle\_t} type. Timer has a callback function associated with it. This callback function is called from the \textbf{esp\_timer} task each time the timer elapses.

- To create a timer, call \textbf{esp\_timer\_create()}.
Chapter 2. API

- To delete the timer when it is no longer needed, call `esp_timer_delete()`.

The timer can be started in one-shot mode or in periodic mode.

- To start the timer in one-shot mode, call `esp_timer_start_once()`, passing the time interval after which the callback should be called. When the callback gets called, the timer is considered to be stopped.
- To start the timer in periodic mode, call `esp_timer_start_periodic()`, passing the period with which the callback should be called. The timer keeps running until `esp_timer_stop()` is called.

Note that the timer must not be running when `esp_timer_start_once()` or `esp_timer_start_periodic()` is called. To restart a running timer, call `esp_timer_stop()` first, then call one of the start functions.

**Callback functions**

备注：Keep the callback functions as short as possible otherwise it will affect all timers.

Timer callbacks which are processed by ESP_TIMER_ISR method should not call the context switch call - `portYIELD_FROM_ISR()`, instead of this you should use the `esp_timer_isr_dispatch_need_yield()` function. The context switch will be done after all ISR dispatch timers have been processed, if required by the system.

**esp_timer during the light sleep**

During light sleep, the esp_timer counter stops and no callback functions are called. Instead, the time is counted by the RTC counter. Upon waking up, the system gets the difference between the counters and calls a function that advances the esp_timer counter. Since the counter has been advanced, the system starts calling callbacks that were not called during sleep. The number of callbacks depends on the duration of the sleep and the period of the timers. It can lead to overflow of some queues. This only applies to periodic timers, one-shot timers will be called once.

This behavior can be changed by calling `esp_timer_stop()` before sleeping. In some cases, this can be inconvenient, and instead of the stop function, you can use the `skip_unhandled_events` option during `esp_timer_create()`. When the `skip_unhandled_events` is true, if a periodic timer expires one or more times during light sleep then only one callback is called on wake.

Using the `skip_unhandled_events` option with automatic light sleep (see Power Management APIs) helps to reduce the consumption of the system when it is in light sleep. The duration of light sleep is also determined by esp_timers. Timers with `skip_unhandled_events` option will not wake up the system.

**Handling callbacks**

esp_timer is designed to achieve a high-resolution low latency timer and the ability to handle delayed events. If the timer is late then the callback will be called as soon as possible, it will not be lost. In the worst case, when the timer has not been processed for more than one period (for periodic timers), in this case the callbacks will be called one after the other without waiting for the set period. This can be bad for some applications, and the `skip_unhandled_events` option was introduced to eliminate this behavior. If `skip_unhandled_events` is set then a periodic timer that has expired multiple times without being able to call the callback will still result in only one callback event once processing is possible.

**Obtaining Current Time**

`esp_timer` also provides a convenience function to obtain the time passed since start-up, with microsecond precision: `esp_timer_get_time()`. This function returns the number of microseconds since `esp_timer` was initialized, which usually happens shortly before `app_main` function is called.

Unlike `gettimeofday` function, values returned by `esp_timer_get_time()`:

Submit Document Feedback
• Start from zero after the chip wakes up from deep sleep
• Do not have timezone or DST adjustments applied

Application Example

The following example illustrates usage of esp_timer APIs: system/esp_timer.

API Reference

Header File

• components/esp_timer/include/esp_timer.h

Functions

**esp_err_t esp_timer_early_init (void)**

Minimal initialization of esp_timer.

This function can be called very early in startup process, after this call only esp_timer_get_time function can be used.

备注: This function is called from startup code. Applications do not need to call this function before using other esp_timer APIs.

返回

• ESP_OK on success

**esp_err_t esp_timer_init (void)**

Initialize esp_timer library.

备注: This function is called from startup code. Applications do not need to call this function before using other esp_timer APIs. Before calling this function, esp_timer_early_init must be called by the startup code.

返回

• ESP_OK on success
• ESP_ERR_NO_MEM if allocation has failed
• ESP_ERR_INVALID_STATE if already initialized
• other errors from interrupt allocator

**esp_err_t esp_timer_deinit (void)**

De-initialize esp_timer library.

备注: Normally this function should not be called from applications

返回

• ESP_OK on success
• ESP_ERR_INVALID_STATE if not yet initialized
**esp_err_t esp_timer_create** (const esp_timer_create_args_t *create_args, esp_timer_handle_t *out_handle)

Create an esp_timer instance.

备注：When done using the timer, delete it with esp_timer_delete function.

参数
- **create_args** – Pointer to a structure with timer creation arguments. Not saved by the library, can be allocated on the stack.
- **out_handle** – [out] Output, pointer to esp_timer_handle_t variable which will hold the created timer handle.

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if some of the create_args are not valid
- ESP_ERR_INVALID_STATE if esp_timer library is not initialized yet
- ESP_ERR_NO_MEM if memory allocation fails

**esp_err_t esp_timer_start_once** (esp_timer_handle_t timer, uint64_t timeout_us)

Start one-shot timer.

参数
- **timer** – timer handle created using esp_timer_create
- **timeout_us** – timer timeout, in microseconds relative to the current moment

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is already running

**esp_err_t esp_timer_start_periodic** (esp_timer_handle_t timer, uint64_t period)

Start a periodic timer.

Timer should not be running when this function is called. This function will start the timer which will trigger every ‘period’ microseconds.

参数
- **timer** – timer handle created using esp_timer_create
- **period** – timer period, in microseconds

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is already running

**esp_err_t esp_timer_restart** (esp_timer_handle_t timer, uint64_t timeout_us)

Restart a currently running timer.

If the given timer is a one-shot timer, the timer is restarted immediately and will timeout once in timeout_us microseconds. If the given timer is a periodic timer, the timer is restarted immediately with a new period of timeout_us microseconds.

参数
- **timer** – timer Handle created using esp_timer_create
- **timeout_us** – Timeout, in microseconds relative to the current time. In case of a periodic timer, also represents the new period.

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is not running
### Chapter 2. API 参考

**esp_err_t esp_timer_stop(esp_timer_handle_t timer)**

Stop the timer.

This function stops the timer previously started using esp_timer_start_once or esp_timer_start_periodic.

- **参数** timer — timer handle created using esp_timer_create
- **返回**
  - ESP_OK on success
  - ESP_ERR_INVALID_STATE if the timer is not running

**esp_err_t esp_timer_delete(esp_timer_handle_t timer)**

Delete an esp_timer instance.

The timer must be stopped before deleting. A one-shot timer which has expired does not need to be stopped.

- **参数** timer — timer handle allocated using esp_timer_create
- **返回**
  - ESP_OK on success
  - ESP_ERR_INVALID_STATE if the timer is running

**int64_t esp_timer_get_time(void)**

Get time in microseconds since boot.

- **返回** number of microseconds since underlying timer has been started

**int64_t esp_timer_get_next_alarm(void)**

Get the timestamp when the next timeout is expected to occur.

- **返回** Timestamp of the nearest timer event, in microseconds. The timebase is the same as for the values returned by esp_timer_get_time.

**int64_t esp_timer_get_next_alarm_for_wake_up(void)**

Get the timestamp when the next timeout is expected to occur skipping those which have skip_unhandled_events flag.

- **返回** Timestamp of the nearest timer event, in microseconds. The timebase is the same as for the values returned by esp_timer_get_time.

**esp_err_t esp_timer_get_period(esp_timer_handle_t timer, uint64_t *period)**

Get the period of a timer.

This function fetches the timeout period of a timer.

---

**备注：** The timeout period is the time interval with which a timer restarts after expiry. For one-shot timers, the period is 0 as there is no periodicity associated with such timers.

- **参数**
  - timer — timer handle allocated using esp_timer_create
  - period — memory to store the timer period value in microseconds
- **返回**
  - ESP_OK on success
  - ESP_ERR_INVALID_ARG if the arguments are invalid

**esp_err_t esp_timer_get_expiry_time(esp_timer_handle_t timer, uint64_t *expiry)**

Get the expiry time of a one-shot timer.

This function fetches the expiry time of a one-shot timer.

---

**备注：** This API returns a valid expiry time only for a one-shot timer. It returns an error if the timer handle passed to the function is for a periodic timer.
### Chapter 2. API 参考

#### 参数
- **timer** - timer handle allocated using esp_timer_create
- **expiry** - memory to store the timeout value in microseconds

#### 返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the arguments are invalid
- ESP_ERR_NOT_SUPPORTED if the timer type is periodic

---

**esp_err_t esp_timer_dump** (FILE *stream)

Dump the list of timers to a stream.

If CONFIG_ESP_TIMER_PROFILING option is enabled, this prints the list of all the existing timers. Otherwise, only the list active timers is printed.

The format is:

```
name period alarm times_armed times_triggered total_callback_run_time
```

where:

- `name` — timer name (if CONFIG_ESP_TIMER_PROFILING is defined), or timer pointer
- `period` — period of timer, in microseconds, or 0 for one-shot timer
- `alarm time` — time of the next alarm, in microseconds since boot, or 0 if the timer is not started

The following fields are printed if CONFIG_ESP_TIMER_PROFILING is defined:

- `times_armed` — number of times the timer was armed via esp_timer_start_X
- `times_triggered` — number of times the callback was called
- `total_callback_run_time` — total time taken by callback to execute, across all calls

#### 参数
- **stream** - stream (such as stdout) to dump the information to

#### 返回
- ESP_OK on success
- ESP_ERR_NO_MEM if can not allocate temporary buffer for the output

---

**void esp_timer_isr_dispatch_need_yield** (void)

Requests a context switch from a timer callback function.

This only works for a timer that has an ISR dispatch method. The context switch will be called after all ISR dispatch timers have been processed.

---

**bool esp_timer_is_active** (esp_timer_handle_t timer)

Returns status of a timer, active or not.

This function is used to identify if the timer is still active or not.

#### 参数
- **timer** — timer handle created using esp_timer_create

#### 返回
- 1 if timer is still active
- 0 if timer is not active.

---

**esp_err_t esp_timer_new_etm_alarm_event** (esp_etm_event_handle_t *out_event)

Get the ETM event handle of esp_timer underlying alarm event.

#### 备注: The created ETM event object can be deleted later by calling esp_etm_del_event

---

**备注: The ETM event is generated by the underlying hardware & quot; systimer, therefore, if the esp_timer
is not clocked by systimer, then no ETM event will be generated.**

#### 参数
- **out_event** — [out] Returned ETM event handle

#### 返回
- ESP_OK Success
• ESP_ERR_INVALID_ARG Parameter error

**Structures**

```c
struct esp_timer_create_args_t
```

Timer configuration passed to `esp_timer_create`.

**Public Members**

```c
esp_timer_cb_t callback
```

Function to call when timer expires.

```c
void *arg
```

Argument to pass to the callback.

```c
esp_timer_dispatch_t dispatch_method
```

Call the callback from task or from ISR.

```c
const char *name
```

Timer name, used in `esp_timer_dump` function.

```c
bool skip_unhandled_events
```

Skip unhandled events for periodic timers.

**Type Definitions**

```c
typedef struct esp_timer *esp_timer_handle_t
```

Opaque type representing a single `esp_timer`.

```c
typedef void (*esp_timer_cb_t)(void *arg)
```

Timer callback function type.

```c
Param arg  pointer to opaque user-specific data
```

**Enumerations**

```c
enum esp_timer_dispatch_t
```

Method for dispatching timer callback.

**Values:**

```c
enumerator ESP_TIMER_TASK
```

Callback is called from timer task.

```c
enumerator ESP_TIMER_MAX
```

Count of the methods for dispatching timer callback.

## 2.10.15 Internal and Unstable APIs

This section is listing some APIs that are internal or likely to be changed or removed in the next releases of ESP-IDF.
Chapter 2. API 参考

API Reference

Header File

- components/esp_rom/include/esp_rom_sys.h

Functions

void esp_rom_software_reset_system (void)

Software Reset digital core include RTC.

It is not recommended to use this function in esp-idf, use esp_restart() instead.

void esp_rom_software_reset_cpu (int cpu_no)

Software Reset cpu core.

It is not recommended to use this function in esp-idf, use esp_restart() instead.

参数 cpu_no:	The CPU to reset, 0 for PRO CPU, 1 for APP CPU.

int esp_rom_printf (const char *fmt, ...)

Print formatted string to console device.

备注: float and long long data are not supported!

参数

- fmt: Format string
- ...: Additional arguments, depending on the format string

返回: int: Total number of characters written on success; A negative number on failure.

void esp_rom_delay_us (uint32_t us)

Pauses execution for us microseconds.

参数 us: Number of microseconds to pause

void esp_rom_install_channel_putc (int channel, void (*putc)(char c))

esp_rom_printf can print message to different channels simultaneously. This function can help install the low
level putc function for esp_rom_printf.

参数

- channel: Channel number (starting from 1)
- putc: Function pointer to the putc implementation. Set NULL can disconnect
esp_rom_printf with putc.

void esp_rom_install_uart_printf (void)

Install UART1 as the default console channel, equivalent to esp_rom_install_channel_putc(1,
esp_rom_uart_putc)

soc_reset_reason_t esp_rom_get_reset_reason (int cpu_no)

Get reset reason of CPU.

参数 cpu_no: CPU number

返回: Reset reason code (see in soc/reset_reasons.h)

void esp_rom_route_intr_matrix (int cpu_core, uint32_t periph_intr_id, uint32_t cpu_intr_num)

Route peripheral interrupt sources to CPU’s interrupt port by matrix.

Usually there’re 4 steps to use an interrupt:

a. Route peripheral interrupt source to CPU. e.g. esp_rom_route_intr_matrix(0,
ETS_WIFI_MAC_INTR_SOURCE, ETS_WMAC_INUM)

b. Set interrupt handler for CPU

c. Enable CPU interrupt
Chapter 2. API 参考

d. Enable peripheral interrupt

参数
- cpu_core - The CPU number, which the peripheral interrupt will inform to
- periph_intr_id - The peripheral interrupt source number
- cpu_intr_num - The CPU interrupt number

uint32_t esp_rom_get_cpu_ticks_per_us (void)
Get the real CPU ticks per us.

2.10.16 Inter-Processor Call

备注： The IPC is an Inter-Processor Call and NOT Inter-Process Communication as found on other operating systems.

Overview

Due to the dual core nature of the ESP32, there are instances where a certain callback must be run in the context of a particular CPU such as:
- When allocating an ISR to an interrupt source of a particular CPU (applies to freeing a particular CPU’s interrupt source as well).
- On particular chips (such as the ESP32), accessing memory that is exclusive to a particular CPU (such as RTC Fast Memory).
- Reading the registers/state of another CPU.

The IPC (Inter-Processor Call) feature allows a particular CPU (the calling CPU) to trigger the execution of a callback function on another CPU (the target CPU). The IPC feature allows execution of a callback function on the target CPU in either a task context, or a High Priority Interrupt context (see High-Level Interrupts for more details). Depending on the context that the callback function is executed in, different restrictions apply to the implementation of the callback function.

IPC in Task Context

The IPC feature implements callback execution in a task context by creating an IPC task for each CPU during application startup. When the calling CPU needs to execute a callback on the target CPU, the callback will execute in the context of the target CPU’s IPC task.

When using IPCs in a task context, users need to consider the following:
- IPC callbacks should ideally be simple and short. An IPC callback should avoid attempting to block or yield.
- The IPC tasks are created at the highest possible priority (i.e., configMAX_PRIORITIES - 1) thus the callback should also run at that priority as a result. However, CONFIG_ESP_IPC_USES_CALLERS_PRIORITY is enabled by default which will temporarily lower the priority of the target CPU’s IPC task to the calling CPU before executing the callback.
- Depending on the complexity of the callback, users may need to configure the stack size of the IPC task via CONFIG_ESP_IPC_TASK_STACK_SIZE.
- The IPC feature is internally protected by a mutex. Therefore, simultaneous IPC calls from two or more calling CPUs will be handled on a first come first serve basis.
**API Usage**  Task Context IPC callbacks have the following restrictions:

- The callback must be of type `void func(void *arg)`
- The callback should avoid attempting to block or yield as this will result in the target CPU’s IPC task blocking or yielding.
- The callback must avoid changing any aspect of the IPC task (e.g., by calling `vTaskPrioritySet(NULL, x)`).

The IPC feature offers the API listed below to execute a callback in a task context on a target CPU. The API allows the calling CPU to block until the callback’s execution has completed, or return immediately once the callback’s execution has started.

- `esp_ipc_call()` will trigger an IPC call on the target CPU. This function will block until the target CPU’s IPC task begins execution of the callback.
- `esp_ipc_call_blocking()` will trigger an IPC on the target CPU. This function will block until the target CPU’s IPC task completes execution of the callback.

**IPC in ISR Context**

In some cases, we need to quickly obtain the state of another CPU such as in a core dump, GDB stub, various unit tests, and DPORT workaround. For such scenarios, the IPC feature supports execution of callbacks in a High Priority Interrupt context. The IPC feature implements the High Priority Interrupt context by reserving a High Priority Interrupt on each CPU for IPC usage. When a calling CPU needs to execute a callback on the target CPU, the callback will execute in the context of the High Priority Interrupt of the target CPU.

When using IPCs in High Priority Interrupt context, users need to consider the following:

- Since the callback is executed in a High Priority Interrupt context, the callback must be written entirely in assembly. See the API Usage below for more details regarding writing assembly callbacks.
- The priority of the reserved High Priority Interrupt is dependent on the `CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL` option
- When the callback executes:
  - The calling CPU will disable interrupts of level 3 and lower
  - Although the priority of the reserved interrupt depends on `CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL`, during the execution of the IPC ISR callback, the target CPU will disable interrupts of level 5 and lower regardless of what `CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL` is set to.

**API Usage**  High Priority Interrupt IPC callbacks have the following restrictions:

- The callback must be of type `void func(void *arg)` but implemented entirely in assembly
- The callback is invoked via the `CALLX0` instruction with register windowing disabled, thus the callback:
  - Must not call any register window related instructions (e.g., entry and retw).
  - Must not call other C functions as register windowing is disabled
- The callback should be placed in IRAM at a 4-byte aligned address
- (On invocation of/after returning from) the callback, the registers `a2, a3, a4` are (saved/restored) automatically thus:
  - `a2` will contain the `void *arg` of the callback
  - `a3/a4` are free to use as scratch registers

The IPC feature offers the API listed below to execute a callback in a High Priority Interrupt context.

- `esp_ipc_isr_asm_call()` will trigger an IPC call on the target CPU. This function will busy-wait until the target CPU begins execution of the callback.
- `esp_ipc_isr_asm_call_blocking()` will trigger an IPC call on the target CPU. This function will busy-wait until the target CPU completes execution of the callback.

The following code-blocks demonstrates a High Priority Interrupt IPC callback written in assembly that simply reads the target CPU’s cycle count.
Chapter 2. API 参考

```c
/* esp_test_ipc_isr_get_cycle_count_other_cpu(void *arg) */
// this function reads CCOUNT of the target CPU and stores it in arg.
// use only a2, a3 and a4 regs here.
.section .iram1, "ax"
.align 4
.global esp_test_ipc_isr_get_cycle_count_other_cpu
.type esp_test_ipc_isr_get_cycle_count_other_cpu, @function
// Args:
// a2 - void* arg
esp_test_ipc_isr_get_cycle_count_other_cpu:
rsr.ccount a3
s32i a3, a2, 0
ret

unit32_t cycle_count;
esp_ipc_isr_asm_call_blocking(esp_test_ipc_isr_get_cycle_count_other_cpu, (void*)cycle_count);
```

**备注:** The number of scratch registers available for use is sufficient for most simple use cases. But if your callback requires more scratch registers, `void *arg` can point to a buffer that is used as a register save area. The callback can then save and restore more registers. See the `system/ipc/ipc_isr`.

**备注:** For more examples of High Priority Interrupt IPC callbacks, see `components/esp_system/port/arch/xtensa/esp_ipc_isr_routines.S` and `components/esp_system/test/test_ipc_isr.S`

The High Priority Interrupt IPC API also provides the following convenience functions that can stall/resume the target CPU. These API utilize the High Priority Interrupt IPC, but supply their own internal callbacks:

- `esp_ipc_isr_stall_other_cpu()` stalls the target CPU. The calling CPU disables interrupts of level 3 and lower while the target CPU will busy-wait with interrupts of level 5 and lower disabled. The target CPU will busy-wait until `esp_ipc_isr_release_other_cpu()` is called.
- `esp_ipc_isr_release_other_cpu()` resumes the target CPU.

### API Reference

#### Header File

- `components/esp_system/include/esp_ipc.h`

#### Functions

- **esp_err_t esp_ipc_call(uint32_t cpu_id, esp_ipc_func_t func, void *arg)**

  Execute a callback on a given CPU.

  Execute a given callback on a particular CPU. The callback must be of type “esp_ipc_func_t” and will be invoked in the context of the target CPU’s IPC task.

  - This function will block the target CPU’s IPC task has begun execution of the callback
  - If another IPC call is ongoing, this function will block until the ongoing IPC call completes
  - The stack size of the IPC task can be configured via the CONFIG_ESP_IPC_TASK_STACK_SIZE option

**备注:** In single-core mode, returns ESP_ERR_INVALID_ARG for cpu_id 1.
### Chapter 2. API 参考

#### 参数
- `cpu_id` - [in] CPU where the given function should be executed (0 or 1)
- `func` - [in] Pointer to a function of type `void func(void* arg)` to be executed
- `arg` - [in] Arbitrary argument of type `void*` to be passed into the function

#### 返回
- `ESP_ERR_INVALID_ARG` if `cpu_id` is invalid
- `ESP_ERR_INVALID_STATE` if the FreeRTOS scheduler is not running
- `ESP_OK` otherwise

```c
esp_err_t esp_ipc_call_blocking(uint32_t cpu_id, esp_ipc_func_t func, void *arg)
```

Execute a callback on a given CPU until and block until it completes.

This function is identical to `esp_ipc_call()` except that this function will block until the execution of the callback completes.

---

**备注:** In single-core mode, returns `ESP_ERR_INVALID_ARG` for `cpu_id 1`.

---

#### Type Definitions

```c
typedef void (*esp_ipc_func_t)(void *arg)
```

IPC Callback.

A callback of this type should be provided as an argument when calling `esp_ipc_call()` or `esp_ipc_call_blocking()`.

---

#### Header File

- `components/esp_system/include/esp_ipc_isr.h`

---

#### Functions

```c
void esp_ipc_isr_asm_call(esp_ipc_isr_func_t func, void *arg)
```

Execute an assembly callback on the other CPU.

Execute a given callback on the other CPU in the context of a High Priority Interrupt.

- This function will busy-wait in a critical section until the other CPU has started execution of the callback
- The callback must be written in assembly, is invoked using a `CALLX0` instruction, and has `a2, a3, a4` as scratch registers. See docs for more details

**备注:** This function is not available in single-core mode.

---

#### 参数
- `func` - [in] Pointer to a function of type `void func(void* arg)` to be executed
- `arg` - [in] Arbitrary argument of type `void*` to be passed into the function

---
void **esp_ipc_isr_asm_call_blocking** (*esp_isr_func_t* func, void *arg)

Execute an assembly callback on the other CPU and busy-wait until it completes.

This function is identical to esp_ipc_isr_asm_call() except that this function will busy-wait until the execution of the callback completes.

**参数**

- **func** - [in] Pointer to a function of type void(func(void* arg)) to be executed
- **arg** - [in] Arbitrary argument of type void* to be passed into the function

void **esp_ipc_isr_stall_other_cpu** (void)

Stall the other CPU.

This function will stall the other CPU. The other CPU is stalled by busy-waiting in the context of a High Priority Interrupt. The other CPU will not be resumed until esp_ipc_isr_release_other_cpu() is called.

**备注:** This function is internally implemented using IPC ISR

**备注:** This function is used for DPORT workaround.

**备注:** If the stall feature is paused using esp_ipc_isr_stall_pause(), this function will have no effect

void **esp_ipc_isr_release_other_cpu** (void)

Release the other CPU.

This function will release the other CPU that was previously stalled from calling esp_ipc_isr_stall_other_cpu().

**备注:** This function is internally implemented using IPC ISR

**备注:** This function is used for DPORT workaround.

**备注:** If the stall feature is paused using esp_ipc_isr_stall_pause(), this function will have no effect

void **esp_ipc_isr_stall_pause** (void)

Pause the CPU stall feature.

This function will pause the CPU stall feature. Once paused, calls to esp_ipc_isr_stall_other_cpu() and esp_ipc_isr_release_other_cpu() will have no effect. If a IPC ISR call is already in progress, this function will busy-wait until the call completes before pausing the CPU stall feature.

void **esp_ipc_isr_stall_abort** (void)

Abort a CPU stall.

This function will abort any stalling routine of the other CPU due to a pervious call to esp_ipc_isr_stall_other_cpu(). This function aborts the stall in a non-recoverable manner, thus should only be called in case of a panic.

**备注:** This function is used in panic handling code
void esp_ipc_isr_stall_resume(void)

  Resume the CPU stall feature.
  
  This function will resume the CPU stall feature that was previously paused by calling esp_ipc_isr_stall_pause(). Once resumed, calls to esp_ipc_isr_stall_other_cpu() and esp_ipc_isr_release_other_cpu() will have effect again.

Type Definitions

typedef void (*esp_ipc_isr_func_t)(void *arg)

  IPC ISR Callback.
  
  A callback of this type should be provided as an argument when calling esp_ipc_isr_asm_call() or esp_ipc_isr_asm_call_blocking().

### 2.10.17 Interrupt allocation

**Overview**

The ESP32 has two cores, with 32 interrupts each. Each interrupt has a certain priority level, most (but not all) interrupts are connected to the interrupt mux.

Because there are more interrupt sources than interrupts, sometimes it makes sense to share an interrupt in multiple drivers. The esp_intr_alloc() abstraction exists to hide all these implementation details.

A driver can allocate an interrupt for a certain peripheral by calling esp_intr_alloc() (or esp_intr_alloc_intrstatus()). It can use the flags passed to this function to set the type of interrupt allocated, specifying a particular level or trigger method. The interrupt allocation code will then find an applicable interrupt, use the interrupt mux to hook it up to the peripheral, and install the given interrupt handler and ISR to it.

This code presents two different types of interrupts, handled differently: shared interrupts and non-shared interrupts. The simplest ones are non-shared interrupts: a separate interrupt is allocated per esp_intr_alloc() call and this interrupt is solely used for the peripheral attached to it, with only one ISR that will get called. On the other hand, shared interrupts can have multiple peripherals triggering them, with multiple ISRs being called when one of the peripherals attached signals an interrupt. Thus, ISRs that are intended for shared interrupts should check the interrupt status of the peripheral they service in order to check if any action is required.

Non-shared interrupts can be either level- or edge-triggered. Shared interrupts can only be level interrupts due to the chance of missed interrupts when edge interrupts are used.

For example, let’s say DevA and DevB share an interrupt. DevB signals an interrupt, so INT line goes high. The ISR handler calls code for DevA but does nothing. Then, ISR handler calls code for DevB, but while doing that, DevA signals an interrupt. DevB’s ISR is done, it clears interrupt status for DevB and exits interrupt code. Now, an interrupt for DevA is still pending, but because the INT line never went low, as DevA kept it high even when the interrupt for DevB was cleared, the interrupt is never serviced.

**Multicore issues**

Peripherals that can generate interrupts can be divided in two types:

- External peripherals, within the ESP32 but outside the Xtensa cores themselves. Most ESP32 peripherals are of this type.
- Internal peripherals, part of the Xtensa CPU cores themselves.

Interrupt handling differs slightly between these two types of peripherals.
**Internal peripheral interrupts** Each Xtensa CPU core has its own set of six internal peripherals:

- Three timer comparators
- A performance monitor
- Two software interrupts.

Internal interrupt sources are defined in esp_intr_alloc.h as `ETS_INTERNAL_/*_INTR_SOURCE`.

These peripherals can only be configured from the core they are associated with. When generating an interrupt, the interrupt they generate is hard-wired to their associated core; it’s not possible to have, for example, an internal timer comparator of one core generate an interrupt on another core. That is why these sources can only be managed using a task running on that specific core. Internal interrupt sources are still allocatable using `esp_intr_alloc()` as normal, but they cannot be shared and will always have a fixed interrupt level (namely, the one associated in hardware with the peripheral).

**External Peripheral Interrupts** The remaining interrupt sources are from external peripherals. These are defined in soc/soc.h as `ETS_*_INTR_SOURCE`.

Non-internal interrupt slots in both CPU cores are wired to an interrupt multiplexer, which can be used to route any external interrupt source to any of these interrupt slots.

- Allocating an external interrupt will always allocate it on the core that does the allocation.
- Freeing an external interrupt must always happen on the same core it was allocated on.
- Disabling and enabling external interrupts from another core is allowed.
- Multiple external interrupt sources can share an interrupt slot by passing `ESP_INTR_FLAG_SHARED` as a flag to `esp_intr_alloc()`.

Care should be taken when calling `esp_intr_alloc()` from a task which is not pinned to a core. During task switching, these tasks can migrate between cores. Therefore it is impossible to tell which CPU the interrupt is allocated on, which makes it difficult to free the interrupt handle and may also cause debugging difficulties. It is advised to use `xTaskCreatePinnedToCore()` with a specific CoreID argument to create tasks that will allocate interrupts. In the case of internal interrupt sources, this is required.

**IRAM-Safe Interrupt Handlers**

The `ESP_INTR_FLAG_IRAM` flag registers an interrupt handler that always runs from IRAM (and reads all its data from DRAM), and therefore does not need to be disabled during flash erase and write operations.

This is useful for interrupts which need a guaranteed minimum execution latency, as flash write and erase operations can be slow (erasers can take tens or hundreds of milliseconds to complete).

It can also be useful to keep an interrupt handler in IRAM if it is called very frequently, to avoid flash cache misses. Refer to the `SPI flash API documentation` for more details.

**Multiple Handlers Sharing A Source**

Several handlers can be assigned to a same source, given that all handlers are allocated using the `ESP_INTR_FLAG_SHARED` flag. They will all be allocated to the interrupt, which the source is attached to, and called sequentially when the source is active. The handlers can be disabled and freed individually. The source is attached to the interrupt (enabled), if one or more handlers are enabled, otherwise detached. A handler will never be called when disabled, while its source may still be triggered if any one of its handler enabled.

Sources attached to non-shared interrupt do not support this feature.

Though the framework support this feature, you have to use it very carefully. There usually exist two ways to stop an interrupt from being triggered: disable the source or mask peripheral interrupt status. IDF only handles enabling and disabling of the source itself, leaving status and mask bits to be handled by users. **Status bits shall either be masked before the handler responsible for it is disabled, either be masked and then properly handled in another enabled interrupt.** Please note that leaving some status bits unhandled without masking them, while
disabling the handlers for them, will cause the interrupt(s) to be triggered indefinitely, resulting therefore in a system crash.

**API Reference**

**Header File**
- `components/esp_hw_support/include/esp_intr_alloc.h`

**Functions**

- **`esp_intr_mark_shared`** *(int intno, int cpu, bool is_in_iram)*

  Mark an interrupt as a shared interrupt.

  This will mark a certain interrupt on the specified CPU as an interrupt that can be used to hook shared interrupt handlers to.

  参数
  - **intno** – The number of the interrupt (0-31)
  - **cpu** – CPU on which the interrupt should be marked as shared (0 or 1)
  - **is_in_iram** – Shared interrupt is for handlers that reside in IRAM and the int can be left enabled while the flash cache is disabled.

  返回 ESP_ERR_INVALID_ARG if cpu or intno is invalid ESP_OK otherwise

- **`esp_intr_reserve`** *(int intno, int cpu)*

  Reserve an interrupt to be used outside of this framework.

  This will mark a certain interrupt on the specified CPU as reserved, not to be allocated for any reason.

  参数
  - **intno** – The number of the interrupt (0-31)
  - **cpu** – CPU on which the interrupt should be marked as shared (0 or 1)

  返回 ESP_ERR_INVALID_ARG if cpu or intno is invalid ESP_OK otherwise

- **`esp_intr_alloc`** *(int source, int flags, intr_handler_t handler, void *arg, intr_handle_t *ret_handle)*

  Allocate an interrupt with the given parameters.

  This finds an interrupt that matches the restrictions as given in the flags parameter, maps the given interrupt source to it and hooks up the given interrupt handler (with optional argument) as well. If needed, it can return a handle for the interrupt as well.

  The interrupt will always be allocated on the core that runs this function.

  If ESP_INTR_FLAG_IRAM flag is used, and handler address is not in IRAM or RTC_FAST_MEM, then ESP_ERR_INVALID_ARG is returned.

  参数
  - **source** – The interrupt source. One of the ETS_*._INTR_SOURCE interrupt mux sources, as defined in soc/soc.h, or one of the internal ETS_INTERNAL_*._INTR_SOURCE sources as defined in this header.
  - **flags** – An ORed mask of the ESP_INTR_FLAG_* defines. These restrict the choice of interrupts that this routine can choose from. If this value is 0, it will default to allocating a non-shared interrupt of level 1, 2 or 3. If this is ESP_INTR_FLAG_SHARED, it will allocate a shared interrupt of level 1. Setting ESP_INTR_FLAG_INTRDISABLED will return from this function with the interrupt disabled.
  - **handler** – The interrupt handler. Must be NULL when an interrupt of level >3 is requested, because these types of interrupts aren’t C-callable.
  - **arg** – Optional argument for passed to the interrupt handler
  - **ret_handle** – Pointer to an intr_handle_t to store a handle that can later be used to request details or free the interrupt. Can be NULL if no handle is required.

  返回 ESP_ERR_INVALID_ARG if the combination of arguments is invalid.

  ESP_ERR_NOT_FOUND No free interrupt found with the specified flags ESP_OK otherwise
Allocate an interrupt with the given parameters.

This essentially does the same as esp_intr_alloc, but allows specifying a register and mask combo. For shared interrupts, the handler is only called if a read from the specified register, ANDed with the mask, returns non-zero. By passing an interrupt status register address and a fitting mask, this can be used to accelerate interrupt handling in the case a shared interrupt is triggered; by checking the interrupt statuses first, the code can decide which ISRs can be skipped.

### esp_err_t esp_intr_alloc_intrstatus (int source, int flags, uint32_t intrstatusreg, uint32_t intrstatusmask, intr_handler_t handler, void *arg, intr_handle_t *ret_handle)

Allocate an interrupt with the given parameters.

This essentially does the same as esp_intr_alloc, but allows specifying a register and mask combo. For shared interrupts, the handler is only called if a read from the specified register, ANDed with the mask, returns non-zero. By passing an interrupt status register address and a fitting mask, this can be used to accelerate interrupt handling in the case a shared interrupt is triggered; by checking the interrupt statuses first, the code can decide which ISRs can be skipped.

#### Parameters

- **source** - The interrupt source. One of the ETS_*_INTR_SOURCE interrupt mux sources, as defined in soc/soc.h, or one of the internal ETS_INTERNAL_*_INTR_SOURCE sources as defined in this header.
- **flags** - An ORed mask of the ESP_INTR_FLAG_* defines. These restrict the choice of interrupts that this routine can choose from. If this value is 0, it will default to allocating a non-shared interrupt of level 1, 2 or 3. If this is ESP_INTR_FLAG_SHARED, it will allocate a shared interrupt of level 1. Setting ESP_INTR_FLAG_INTRDISABLED will return from this function with the interrupt disabled.
- **intrstatusreg** - The address of an interrupt status register
- **intrstatusmask** - A mask. If a read of address intrstatusreg has any of the bits that are 1 in the mask set, the ISR will be called. If not, it will be skipped.
- **handler** - The interrupt handler. Must be NULL when an interrupt of level >3 is requested, because these types of interrupts aren’t C-callable.
- **arg** - Optional argument for passed to the interrupt handler
- **ret_handle** - Pointer to an intr_handle_t to store a handle that can later be used to request details or free the interrupt. Can be NULL if no handle is required.

#### Returns

- ESP_ERR_INVALID_ARG if the combination of arguments is invalid.
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags ESP_OK otherwise

### esp_err_t esp_intr_free (intr_handle_t handle)

Disable and free an interrupt.

Use an interrupt handle to disable the interrupt and release the resources associated with it. If the current core is not the core that registered this interrupt, this routine will be assigned to the core that allocated this interrupt, blocking and waiting until the resource is successfully released.

### esp_err_t esp_intr_get_cpu (intr_handle_t handle)

Get CPU number an interrupt is tied to.

#### Parameters

- **handle** - The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

#### Returns

- ESP_ERR_INVALID_ARG the handle is NULL ESP_FAIL failed to release this handle ESP_OK otherwise

### int esp_intr_get_intno (intr_handle_t handle)

Get the allocated interrupt for a certain handle.

#### Parameters

- **handle** - The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

#### Returns

- The interrupt number
**esp_err_t esp_intr_disable (intr_handle_t handle)**

Disable the interrupt associated with the handle.

**备注:**

a. For local interrupts (ESP_INTERNAL_* sources), this function has to be called on the CPU the interrupt is allocated on. Other interrupts have no such restriction.

b. When several handlers sharing a same interrupt source, interrupt status bits, which are handled in the handler to be disabled, should be masked before the disabling, or handled in other enabled interrupts properly. Miss of interrupt status handling will cause infinite interrupt calls and finally system crash.

**参数 handle** - The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

**返回** ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

**esp_err_t esp_intr_enable (intr_handle_t handle)**

Enable the interrupt associated with the handle.

**备注:** For local interrupts (ESP_INTERNAL_* sources), this function has to be called on the CPU the interrupt is allocated on. Other interrupts have no such restriction.

**参数 handle** - The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

**返回** ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

**esp_err_t esp_intr_set_in_iram (intr_handle_t handle, bool is_in_iram)**

Set the “in IRAM” status of the handler.

**备注:** Does not work on shared interrupts.

**参数**

- **handle** - The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus
- **is_in_iram** - Whether the handler associated with this handle resides in IRAM. Handlers residing in IRAM can be called when cache is disabled.

**返回** ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

**void esp_intr_noniram_disable (void)**

Disable interrupts that aren’t specifically marked as running from IRAM.

**void esp_intr_noniram_enable (void)**

Re-enable interrupts disabled by esp_intr_noniram_disable.

**void esp_intr_enable_source (int inum)**

enable the interrupt source based on its number

**参数** inum - interrupt number from 0 to 31

**void esp_intr_disable_source (int inum)**

disable the interrupt source based on its number

**参数** inum - interrupt number from 0 to 31

**static inline int esp_intr_flags_to_level (int flags)**

Get the lowest interrupt level from the flags.

**参数** flags - The same flags that pass to esp_intr_alloc_intrstatus API
Macros

ESP_INTR_FLAG_LEVEL1
Interrupt allocation flags.
These flags can be used to specify which interrupt qualities the code calling esp_intr_alloc* needs. Accept a Level 1 interrupt vector (lowest priority)

ESP_INTR_FLAG_LEVEL2
Accept a Level 2 interrupt vector.

ESP_INTR_FLAG_LEVEL3
Accept a Level 3 interrupt vector.

ESP_INTR_FLAG_LEVEL4
Accept a Level 4 interrupt vector.

ESP_INTR_FLAG_LEVEL5
Accept a Level 5 interrupt vector.

ESP_INTR_FLAG_LEVEL6
Accept a Level 6 interrupt vector.

ESP_INTR_FLAG_NMI
Accept a Level 7 interrupt vector (highest priority)

ESP_INTR_FLAG_SHARED
Interrupt can be shared between ISRs.

ESP_INTR_FLAG_EDGE
Edge-triggered interrupt.

ESP_INTR_FLAG_IRAM
ISR can be called if cache is disabled.

ESP_INTR_FLAG_INTRDISABLED
Return with this interrupt disabled.

ESP_INTR_FLAG_LOWMED
Low and medium prio interrupts. These can be handled in C.

ESP_INTR_FLAG_HIGH
High level interrupts. Need to be handled in assembly.

ESP_INTR_FLAG_LEVELMASK
Mask for all level flags.

ETS_INTERNAL_TIMER0_INTR_SOURCE
Platform timer 0 interrupt source.
The esp_intr_alloc* functions can allocate an int for all ETS_*_INTR_SOURCE interrupt sources that are routed through the interrupt mux. Apart from these sources, each core also has some internal sources that do
not pass through the interrupt mux. To allocate an interrupt for these sources, pass these pseudo-sources to the functions.

**ETS_INTERNAL_TIMER1_INTR_SOURCE**
Platform timer 1 interrupt source.

**ETS_INTERNAL_TIMER2_INTR_SOURCE**
Platform timer 2 interrupt source.

**ETS_INTERNAL_SW0_INTR_SOURCE**
Software int source 1.

**ETS_INTERNAL_SW1_INTR_SOURCE**
Software int source 2.

**ETS_INTERNAL_PROFILING_INTR_SOURCE**
Int source for profiling.

**ETS_INTERNAL_UNUSED_INTR_SOURCE**
Interrupt is not assigned to any source.

**ETS_INTERNAL_INTR_SOURCE_OFF**
Provides SystemView with positive IRQ IDs, otherwise scheduler events are not shown properly.

**ESP_INTR_ENABLE** (inum)
Enable interrupt by interrupt number.

**ESP_INTR_DISABLE** (inum)
Disable interrupt by interrupt number.

**Type Definitions**

typedef void (*intr_handler_t)(void *arg)
Function prototype for interrupt handler function.

typedef struct intr_handle_data_t intr_handle_data_t
Interrupt handler associated data structure.

typedef intr_handle_data_t *intr_handle_t
Handle to an interrupt handler.

### 2.10.18 Logging library

**Overview**

The logging library provides two ways for setting log verbosity:

- **At compile time:** in menuconfig, set the verbosity level using the option `CONFIG_LOG_DEFAULT_LEVEL`. Optionally, also in menuconfig, set the maximum verbosity level using the option `CONFIG_LOG_MAXIMUM_LEVEL`. By default this is the same as the default level, but it can be set higher in order to compile more optional logs into the firmware.
- **At runtime:** all logs for verbosity levels lower than `CONFIG_LOG_DEFAULT_LEVEL` are enabled by default. The function `esp_log_level_set()` can be used to set a logging level on a per module basis. Modules are identified by their tags, which are human-readable ASCII zero-terminated strings.
There are the following verbosity levels:

- Error (lowest)
- Warning
- Info
- Debug
- Verbose (highest)

**How to use this library**

In each C file that uses logging functionality, define the TAG variable as shown below:

```c
static const char* TAG = "MyModule";
```

Then use one of the logging macros to produce output, e.g:

```c
ESP_LOGW(TAG, "Baud rate error %.1f%%. Requested: %d baud, actual: %d baud", error_→* 100, baud_req, baud_real);
```

Several macros are available for different verbosity levels:

- ESP_LOGE - error (lowest)
- ESP_LOGW - warning
- ESP_LOGI - info
- ESP_LOGD - debug
- ESP_LOGV - verbose (highest)

Additionally, there are ESP_EARLY_LOGx versions for each of these macros, e.g. ESP_EARLY_LOGE. These versions have to be used explicitly in the early startup code only, before heap allocator and syscalls have been initialized. Normal ESP_LOGx macros can also be used while compiling the bootloader, but they will fall back to the same implementation as ESP_EARLY_LOGx macros.

There are also ESP_DRAM_LOGx versions for each of these macros, e.g. ESP_DRAM_LOGE. These versions are used in some places where logging may occur with interrupts disabled or with flash cache inaccessible. Use of this macros should be as sparing as possible, as logging in these types of code should be avoided for performance reasons.

**备注：** Inside critical sections interrupts are disabled so it’s only possible to use ESP_DRAM_LOGx (preferred) or ESP_EARLY_LOGx. Even though it’s possible to log in these situations, it’s better if your program can be structured not to require it.

To override default verbosity level at file or component scope, define the LOG_LOCAL_LEVEL macro.

At file scope, define it before including esp_log.h, e.g:

```c
#define LOG_LOCAL_LEVEL ESP_LOG_VERBOSE
#include "esp_log.h"
```

At component scope, define it in the component makefile:

```c
TARGET_COMPILE_DEFINITIONS("${COMPONENT_LIB} PUBLIC "-DLOG_LOCAL_LEVEL=ESP_LOG_→_VERBOSE")
```

To configure logging output per module at runtime, add calls to the function `esp_log_level_set()` as follows:
The DRAM and EARLY log macro variants documented above do not support per module setting of log verbosity. These macros will always log at the “default” verbosity level, which can only be changed at runtime by calling `esp_log_level_set("\", level).

Logging to Host via JTAG

By default, the logging library uses the vprintf-like function to write formatted output to the dedicated UART. By calling a simple API, all log output may be routed to JTAG instead, making logging several times faster. For details, please refer to Section Logging to Host.

Application Example

The logging library is commonly used by most esp-idf components and examples. For demonstration of log functionality, check ESP-IDF’s examples directory. The most relevant examples that deal with logging are the following:

- system/ota
- storage/sd_card
- protocols/https_request

API Reference

Header File

- components/log/include/esp_log.h

Functions

`void esp_log_level_set(const char *tag, esp_log_level_t level)`

Set log level for given tag.

If logging for given component has already been enabled, changes previous setting.

备注: Note that this function can not raise log level above the level set using CONFIG_LOG_MAXIMUM_LEVEL setting in menuconfig. To raise log level above the default one for a given file, define LOG_LOCAL_LEVEL to one of the ESP_LOG_* values, before including esp_log.h in this file.

参数

- `tag` - Tag of the log entries to enable. Must be a non-NULL zero terminated string. Value "\" resets log level for all tags to the given value.
- `level` - Selects log level to enable. Only logs at this and lower verbosity levels will be shown.

`esp_log_level_t esp_log_level_get(const char *tag)`

Get log level for a given tag, can be used to avoid expensive log statements.

备注: The current log level for the given tag.

参数 `tag` - Tag of the log to query current level. Must be a non-NULL zero terminated string.

返回 The current log level for the given tag

`vprintf_like_t esp_log_set_vprintf(vprintf_like_t func)`

Set function used to output log entries.
By default, log output goes to UART0. This function can be used to redirect log output to some other destination, such as file or network. Returns the original log handler, which may be necessary to return output to the previous destination.

Please note that function callback here must be re-entrant as it can be invoked in parallel from multiple thread contexts.

- **func** - new Function used for output. Must have same signature as vprintf.
- **oldFunction** - old Function used for output.

```c
uint32_t esp_log_timestamp (void)
```

Function which returns timestamp to be used in log output.

This function is used in expansion of ESP_LOGx macros. In the 2nd stage bootloader, and at early application startup stage this function uses CPU cycle counter as time source. Later when FreeRTOS scheduler start running, it switches to FreeRTOS tick count.

For now, we ignore millisecond counter overflow.

```c
char * esp_log_system_timestamp (void)
```

Function which returns system timestamp to be used in log output.

This function is used in expansion of ESP_LOGx macros to print the system time as “HH:MM:SS.sss”. The system time is initialized to 0 on startup, this can be set to the correct time with an SNTP sync, or manually with standard POSIX time functions.

Currently, this will not get used in logging from binary blobs (i.e. Wi-Fi & Bluetooth libraries), these will still print the RTOS tick time.

```c
uint32_t esp_log_early_timestamp (void)
```

Function which returns timestamp to be used in log output.

This function uses HW cycle counter and does not depend on OS, so it can be safely used after application crash.

```c
void esp_log_write (esp_log_level_t level, const char *tag, const char *format, ...)
```

Write message into the log.

This function is not intended to be used directly. Instead, use one of ESP_LOGE, ESP_LOGW, ESP_LOGI, ESP_LOGD, ESP_LOGV macros.

This function or these macros should not be used from an interrupt.

```c
void esp_log_writev (esp_log_level_t level, const char *tag, const char *format, va_list args)
```

Write message into the log, va_list variant.

This function is provided to ease integration toward other logging framework, so that esp_log can be used as a log sink.

Espressif Systems 1993

Submit Document Feedback
Chapter 2. API  參考

Macros

**ESP_LOG_BUFFER_HEX_LEVEL** (tag, buffer, buff_len, level)

Log a buffer of hex bytes at specified level, separated into 16 bytes each line.

参数
- **tag** – description tag
- **buffer** – Pointer to the buffer array
- **buff_len** – length of buffer in bytes
- **level** – level of the log

**ESP_LOG_BUFFER_CHAR_LEVEL** (tag, buffer, buff_len, level)

Log a buffer of characters at specified level, separated into 16 bytes each line. Buffer should contain only printable characters.

参数
- **tag** – description tag
- **buffer** – Pointer to the buffer array
- **buff_len** – length of buffer in bytes
- **level** – level of the log

**ESP_LOG_BUFFER_HEXDUMP** (tag, buffer, buff_len, level)

Dump a buffer to the log at specified level.

The dump log shows just like the one below:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W (195) log_example: 0x3ffb4280</td>
<td>45 53 50 33 32 20 69 73 20 67 72 65 61 74...</td>
</tr>
<tr>
<td>→ 2c 20</td>
<td>ESP32 is great,</td>
</tr>
<tr>
<td>W (195) log_example: 0x3ffb4290</td>
<td>77 6f 72 6b 69 6e 67 72 65 61 74 20 61 6c 6f 6e 67 20...</td>
</tr>
<tr>
<td>→ 77 69</td>
<td>working along wi</td>
</tr>
<tr>
<td>W (205) log_example: 0x3ffb42a0</td>
<td>74 68 20 74 68 65 20 49 44 46 2e 00 44 4e 4e 72 69 6e 67 20 61 6c 6f 6e 67 20...</td>
</tr>
<tr>
<td>→</td>
<td>th the IDF.</td>
</tr>
</tbody>
</table>

It is highly recommended to use terminals with over 102 text width.

参数
- **tag** – description tag
- **buffer** – Pointer to the buffer array
- **buff_len** – length of buffer in bytes
- **level** – level of the log

**ESP_LOG_BUFFER_HEX** (tag, buffer, buff_len)

Log a buffer of hex bytes at Info level.

參見:

esp_log_buffer_hex_level

参数
- **tag** – description tag
- **buffer** – Pointer to the buffer array
- **buff_len** – length of buffer in bytes

**ESP_LOG_BUFFER_CHAR** (tag, buffer, buff_len)

Log a buffer of characters at Info level. Buffer should contain only printable characters.

參見:

esp_log_buffer_char_level

参数
- **tag** – description tag
Chapter 2. API 参考

- **buffer** - Pointer to the buffer array
- **buff_len** - length of buffer in bytes

**ESP_EARLY_LOGE** (tag, format, ...)

Macro to output logs in startup code, before heap allocator and syscalls have been initialized. Log at ESP_LOG_ERROR level.

```c
ESP_EARLY_LOGE(tag, format, ...)
```

参见:

`printf, ESP_LOGE, ESP_DRAM_LOGE`

In the future, we want to switch to C++20. We also want to become compatible with clang. Hence, we provide two versions of the following macros which are using variadic arguments. The first one is using the GNU extension `##__VA_ARGS__`. The second one is using the C++20 feature `VA_OPT(,)`. This allows users to compile their code with standard C++20 enabled instead of the GNU extension. Below C++20, we haven’t found any good alternative to using `##__VA_ARGS__`.

**ESP_EARLY_LOGW** (tag, format, ...)

Macro to output logs in startup code at ESP_LOG_WARN level.

```c
ESP_EARLY_LOGW(tag, format, ...)
```

参见:

`ESP_EARLY_LOGE, ESP_LOGE, printf`

**ESP_EARLY_LOGI** (tag, format, ...)

Macro to output logs in startup code at ESP_LOG_INFO level.

```c
ESP_EARLY_LOGI(tag, format, ...)
```

参见:

`ESP_EARLY_LOGE, ESP_LOGE, printf`

**ESP_EARLY_LOGD** (tag, format, ...)

Macro to output logs in startup code at ESP_LOG_DEBUG level.

```c
ESP_EARLY_LOGD(tag, format, ...)
```

参见:

`ESP_EARLY_LOGE, ESP_LOGE, printf`

**ESP_EARLY_LOGV** (tag, format, ...)

Macro to output logs in startup code at ESP_LOG_VERBOSE level.

```c
ESP_EARLY_LOGV(tag, format, ...)
```

参见:

`ESP_EARLY_LOGE, ESP_LOGE, printf`

**_ESP_LOG_EARLY_ENABLED** (log_level)

**ESP_LOG_EARLY_IMPL** (tag, format, log_level, log_tag_letter, ...)

**ESP_LOGE** (tag, format, ...)

**ESP_LOGW** (tag, format, ...)

**ESP_LOGI** (tag, format, ...)

**ESP_LOGD** (tag, format, ...)

**ESP_LOGV** (tag, format, ...)
**ESP_LOG_LEVEL** (level, tag, format, ...)  
runtime macro to output logs at a specified level.

```
参见:
printf
```

参数:
- **tag** — tag of the log, which can be used to change the log level by *esp_log_level_set* at runtime.
- **level** — level of the output log.
- **format** — format of the output log. See *printf*
- ... — variables to be replaced into the log. See *printf*

**ESP_LOG_LEVEL_LOCAL** (level, tag, format, ...)  
runtime macro to output logs at a specified level. Also check the level with *LOG_LOCAL_LEVEL*.

参见:
printf, ESP_LOG_LEVEL

**ESP_DRAM_LOGE** (tag, format, ...)
Macro to output logs when the cache is disabled. Log at ESP_LOG_ERROR level.

Similar to

Usage:  
*ESP_DRAM_LOGE*(DRAM_STR("my_tag"), "format", orESP_DRAM_LOGE(TAG, "format", ...)); where TAG is a char* that points to a str in the DRAM.

参见:

ESP_EARLY_LOGE, the log level cannot be changed per-tag, however *esp_log_level_set*( "*", level) will set the default level which controls these log lines also.

参见:

esp_rom_printf, ESP_LOGE

备注: Unlike normal logging macros, it’s possible to use this macro when interrupts are disabled or inside an ISR.

备注: Placing log strings in DRAM reduces available DRAM, so only use when absolutely essential.

**ESP_DRAM_LOGW** (tag, format, ...)
macro to output logs when the cache is disabled at ESP_LOG_WARN level.

参见:

ESP_DRAM_LOGW, ESP_LOGW, esp_rom_printf
**Chapter 2. API 參考**

**ESP_DRAM_LOGI** (tag, format,...)  
macro to output logs when the cache is disabled at ESP_LOG_INFO level.

参見: ESP_DRAM_LOGI,ESP_LOGI,esp_rom_printf

**ESP_DRAM_LOGD** (tag, format,...)  
macro to output logs when the cache is disabled at ESP_LOG_DEBUG level.

参見: ESP_DRAM_LOGD,ESP_LOGD,esp_rom_printf

**ESP_DRAM_LOGV** (tag, format,...)  
macro to output logs when the cache is disabled at ESP_LOG_VERBOSE level.

参見: ESP_DRAM_LOGV,ESP_LOGV,esp_rom_printf

**Type Definitions**

typedef int (*vprintf_like_t)(const char*, va_list)

**Enumerations**

cenum esp_log_level_t  
Log level.  
Values:

enumerator ESP_LOG_NONE  
No log output

enumerator ESP_LOG_ERROR  
Critical errors, software module can not recover on its own

enumerator ESP_LOG_WARN  
Error conditions from which recovery measures have been taken

enumerator ESP_LOG_INFO  
Information messages which describe normal flow of events

enumerator ESP_LOG_DEBUG  
Extra information which is not necessary for normal use (values, pointers, sizes, etc).

enumerator ESP_LOG_VERBOSE  
Bigger chunks of debugging information, or frequent messages which can potentially flood the output.

2.10.19 宮項系統 API
## 软件复位

函数 `esp_restart()` 用于执行芯片的软件复位。调用此函数时，程序停止执行，两个 CPU 复位，应用程序由 bootloader 加载并重启。

函数 `esp_register_shutdown_handler()` 用于注册复位前会自动调用的例程（复位过程由 `esp_restart()` 函数触发），这与 `atexit` POSIX 函数的功能类似。

## 复位原因

ESP-IDF 应用程序启动或复位的原因有多种。调用 `esp_reset_reason()` 函数可获取最近一次复位的原因。复位的所有可能原因，请查看 `esp_reset_reason_t` 中的描述。

## 堆内存

ESP-IDF 中有两个与堆内存相关的函数:

- 函数 `esp_get_free_heap_size()` 用于查询当前可用的堆内存大小。
- 函数 `esp_get_minimum_free_heap_size()` 用于查询整个过程中可用的最小堆内存大小（例如应用程序生命周期内可用的最小堆内存大小）。

请注意，ESP-IDF 支持功能不同的的多个堆。上文中函数返回的堆内存大小可使用 `malloc` 函数族来进行分配。有关堆内存的更多信息，请参阅“堆内存分配”。

## MAC 地址

以下 API 用于查询和自定义支持的网络接口（如 Wi-Fi、蓝牙、以太网）的 MAC 地址。

要获取特定接口（如 Wi-Fi、蓝牙、以太网）的 MAC 地址，请调用函数 `esp_read_mac()`。

在 ESP-IDF 中，各个网络接口的 MAC 地址是根据单个 基准 MAC 地址 (Base MAC address) 计算出来的。默认情况下使用乐鑫指定的基准 MAC 地址，该基准地址在产品生产过程中已预烧录至 ESP32 eFuse。

<table>
<thead>
<tr>
<th>接口</th>
<th>MAC 地址 (默认 4 个全局地址)</th>
<th>MAC 地址 (2 个全局地址)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi 站点</td>
<td>base_mac</td>
<td>base_mac</td>
</tr>
<tr>
<td>Wi-Fi SoftAP</td>
<td>base_mac 后一组字节后加1</td>
<td>本地 MAC（由 Wi-Fi 站点 MAC 生成）</td>
</tr>
<tr>
<td>蓝牙</td>
<td>base_mac 后一组字节后加2</td>
<td>base_mac 后一组字节后加1</td>
</tr>
<tr>
<td>以太网</td>
<td>base_mac 后一组字节后加3</td>
<td>本地 MAC（由蓝牙 MAC 生成）</td>
</tr>
</tbody>
</table>

**备注：** 配置选项 配置了乐鑫提供的全局 MAC 地址的数量。

### 自定义基准 MAC

乐鑫已将默认的基准 MAC 地址预烧录至 eFuse BLK0 中。如需设置自定义基准 MAC 地址，请在初始化任一网络接口或调用 `esp_read_mac()` 函数前调用 `esp_base_mac_addr_set()` 函数。自定义基准 MAC 地址可以存储在任何支持的存储设备中（如 flash，NVS）。

分配自定义基 MAC 地址时，应避免 MAC 地址重叠。请根据上面的表格配置选 `option CONFIG_ESP32_UNIVERSAL_MAC_ADDRESSES`，设置可从自定义基 MAC 地址生成的有解放全局 MAC 地址。

**备注：** 也可以调用函数 `esp_netif_set_mac()`，在网络初始化后设置网络接口使用的特定 MAC。但建议使用此处介绍的自定义基准 MAC 地址的方法，以避免原始 MAC 地址在更改前短暂出现在网络上。
**eFuse 中的自定义 MAC 地址**

ESP-IDF 提供了 `esp_efuse_mac_get_custom()` 函数，从 eFuse 读取自定义 MAC 地址时，调用该函数将从 eFuse BLK3 加载 MAC 地址。此函数假设自定义 MAC 地址的存储格式如下:

<table>
<thead>
<tr>
<th>字段</th>
<th>比特数</th>
<th>比特范围</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>8</td>
<td>191:184</td>
<td>0：无效；其他：有效</td>
</tr>
<tr>
<td>Reserved</td>
<td>128</td>
<td>183:56</td>
<td></td>
</tr>
<tr>
<td>MAC address</td>
<td>48</td>
<td>55:8</td>
<td></td>
</tr>
<tr>
<td>MAC address CRC</td>
<td>8</td>
<td>7:0</td>
<td>CRC-8-CCITT，多项式 0x07</td>
</tr>
</tbody>
</table>

**备注：**如果启用了 3/4 编码方案，则必须同时烧写该块中的所有 eFuse 字段。

调用 `esp_efuse_mac_get_custom()` 函数获得 MAC 地址后，请调用 `esp_base_mac_addr_set()` 函数将此 MAC 地址设置为基准 MAC 地址。

**本地 MAC 地址和全局 MAC 地址**

在 ESP32 中，乐鑫已预烧录足够数量的有效乐鑫全局 MAC 地址，供所有内部接口使用。上文中的表格已经介绍了如何根据基准 MAC 地址计算出具体接口的 MAC 地址。

当使用自定义 MAC 地址时，可能并非所有接口都能被分配到一个全局 MAC 地址。此时，接口会被分配一个本地 MAC 地址。请注意，这些地址仅用于单个本地网络。

**本地 MAC 地址和全局 MAC 地址的定义，请参见此处。**

内部调用函数 `esp_derive_local_mac()`，可从全局 MAC 地址生成本地 MAC 地址。具体流程如下：

1. 在全局 MAC 地址的第一个字节组中设置 U/L 位（位值为 0x2），创建本地 MAC 地址。
2. 如果该位已存在于全局 MAC 地址中（即现有的“全局”MAC 地址实际上已经是本地 MAC 地址），则本地 MAC 地址的第一个字节组与 0x4 异或。

**芯片版本**

`esp_chip_info()` 函数用于填充 `esp_chip_info_t` 结构体中的芯片信息，包括芯片版本、CPU 数量和芯片中已启用功能的位掩码。

**SDK 版本**

调用函数 `esp_get_idf_version()` 可返回一个字符串，该字符串包含了用于编译应用程序的 ESP-IDF 版本，与构建系统中通过 IDF_VER 变量所获得的值相同。该版本字符串的格式即 `git describe` 命令的运行结果。

也有其它版本可用于在构建过程中获取 ESP-IDF 版本，它们可根据 ESP-IDF 版本启用或禁用部分程序。

- `ESP_IDF_VERSION_MAJOR, ESP_IDF_VERSION_MINOR` 和 `ESP_IDF_VERSION_PATCH` 分别被定义为代表主版本、次要版本和补丁版本的整数。
- `ESP_IDF_VERSION_VAL` 和 `ESP_IDF_VERSION` 可在确认版本时使用:

```c
#include "esp_idf_version.h"

#if ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)
    // 启用 ESP-IDF v4.0 中的功能
#endif
```
应用程序版本

应用程序版本存储在 esp_app_desc_t 结构体中。该结构体位于 DROM 扇区，有一个从二进制文件头部计算的固定偏移值。该结构体位于 esp_image_header_t 和 esp_image_segment_header_t 结构体之后，字段 Version 类型为字符串，最大长度为 32 字节。

若需手动设置版本，需要在项目的 CMakeLists.txt 文件中设置 PROJECT_VER 变量，即在 CMakeLists.txt 文件中，在包含 project.cmake 之前添加 set(PROJECT_VER "0.1.0.1")。

如果设置了 CONFIG_APP_PROJECT_VER_FROM_CONFIG 选项，则将使用 CONFIG_APP_PROJECT_VER 的值。否则，如果在项目中未设置 PROJECT_VER 变量，则该变量将从 $(PROJECT_PATH)/version.txt 文件（若有）中检索，或使用 git 命令 git describe 检索。如果两者都不可用，则 PROJECT_VER 将被设置为“1”。应用程序可通过调用 esp_app_get_description() 或 esp_ota_get_partition_description() 函数来获取应用程序的版本信息。

API 参考

Header File

- components/esp_system/include/esp_system.h

Functions

* esp_err_t esp_register_shutdown_handler(shutdown_handler_t handle)
  
  Register shutdown handler.
  
  This function allows you to register a handler that gets invoked before the application is restarted using esp_restart function.

  参数 handle -function to execute on restart

  返回
  - ESP_OK on success
  - ESP_ERR_INVALID_STATE if the handler has already been registered
  - ESP_ERR_NO_MEM if no more shutdown handler slots are available

* esp_err_t esp_unregister_shutdown_handler(shutdown_handler_t handle)
  
  Unregister shutdown handler.

  This function allows you to unregister a handler which was previously registered using esp_register_shutdown_handler function.

  参数 handle

  返回
  - ESP_OK on success
  - ESP_ERR_INVALID_STATE if the given handler hasn’t been registered before

* void esp_restart(void)
  
  Restart PRO and APP CPUs.

  This function can be called both from PRO and APP CPUs. After successful restart, CPU reset reason will be SW_CPU_RESET. Peripherals (except for Wi-Fi, BT, UART0, SPI1, and legacy timers) are not reset. This function does not return.

* esp_reset_reason_t esp_reset_reason(void)
  
  Get reason of last reset.

  返回 See description of esp_reset_reason_t for explanation of each value.

* uint32_t esp_get_free_heap_size(void)
  
  Get the size of available heap.

  备注: Note that the returned value may be larger than the maximum contiguous block which can be allocated.

  返回 Available heap size, in bytes.
uint32_t esp_get_free_internal_heap_size (void)
    Get the size of available internal heap.

    备注：Note that the returned value may be larger than the maximum contiguous block which can be allocated.

    返回 Available internal heap size, in bytes.

uint32_t esp_get_minimum_free_heap_size (void)
    Get the minimum heap that has ever been available.

    返回 Minimum free heap ever available

void esp_system_abort (const char *details)
    Trigger a software abort.

    参数 details Details that will be displayed during panic handling.

Type Definitions

typedef void (*shutdown_handler_t)(void)
    Shutdown handler type

Enumerations

enum esp_reset_reason_t
    Reset reasons.

    Values:

    enumerator ESP_RST_UNKNOWN
        Reset reason can not be determined.

    enumerator ESP_RST_POWERON
        Reset due to power-on event.

    enumerator ESP_RST_EXT
        Reset by external pin (not applicable for ESP32)

    enumerator ESP_RST_SW
        Software reset via esp_restart.

    enumerator ESP_RST_PANIC
        Software reset due to exception/panic.

    enumerator ESP_RST_INT_WDT
        Reset (software or hardware) due to interrupt watchdog.

    enumerator ESP_RST_TASK_WDT
        Reset due to task watchdog.

    enumerator ESP_RST_WDT
        Reset due to other watchdogs.
enumerator **ESP_RST_DEEPSLEEP**

Reset after exiting deep sleep mode.

enumerator **ESP_RST_BROWNOUT**

Brownout reset (software or hardware)

enumerator **ESP_RST_SDIO**

Reset over SDIO.

**Header File**

- components/esp_common/include/esp_idf_version.h

**Functions**

const char* `esp_get_idf_version` (void)

Return full IDF version string, same as `git describe` output.

备注：If you are printing the ESP-IDF version in a log file or other information, this function provides more information than using the numerical version macros. For example, numerical version macros don’t differentiate between development, pre-release and release versions, but the output of this function does.

返回 constant string from IDF_VER

**Macros**

**ESP_IDF_VERSION_MAJOR**

Major version number (X.X.X)

**ESP_IDF_VERSION_MINOR**

Minor version number (x.X.x)

**ESP_IDF_VERSION_PATCH**

Patch version number (x.x.X)

**ESP_IDF_VERSION_VAL** (major, minor, patch)

Macro to convert IDF version number into an integer

To be used in comparisons, such as `ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)`

**ESP_IDF_VERSION**

Current IDF version, as an integer

To be used in comparisons, such as `ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)`

**Header File**

- components/esp_hw_support/include/esp_mac.h
Chapter 2. API 参考

**Functions**

esp_err_t esp_base_mac_addr_set (const uint8_t* mac)

Set base MAC address with the MAC address which is stored in BLK3 of EFUSE or external storage e.g. flash and EEPROM.

Base MAC address is used to generate the MAC addresses used by network interfaces.

If using a custom base MAC address, call this API before initializing any network interfaces. Refer to the ESP-IDF Programming Guide for details about how the Base MAC is used.

备注：Base MAC must be a unicast MAC (least significant bit of first byte must be zero).

备注：If not using a valid OUI, set the “locally administered” bit (bit value 0x02 in the first byte) to avoid collisions.

参数 mac – base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4)

返回 ESP_OK on success ESP_ERR_INVALID_ARG If mac is NULL or is not a unicast MAC

esp_err_t esp_base_mac_addr_get (uint8_t* mac)

Return base MAC address which is set using esp_base_mac_addr_set.

备注：If no custom Base MAC has been set, this returns the pre-programmed Espressif base MAC address.

参数 mac – base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4)

返回 ESP_OK on success ESP_ERR_INVALID_ARG mac is NULL

ESP_ERR_INVALID_MAC base MAC address has not been set

esp_err_t esp_efuse_mac_get_custom (uint8_t* mac)

Return base MAC address which was previously written to BLK3 of EFUSE.

Base MAC address is used to generate the MAC addresses used by the networking interfaces. This API returns the custom base MAC address which was previously written to EFUSE BLK3 in a specified format.

Writing this EFUSE allows setting of a different (non-Espressif) base MAC address. It is also possible to store a custom base MAC address elsewhere, see esp_base_mac_addr_set() for details.

备注：This function is currently only supported on ESP32.

参数 mac – base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4)

返回 ESP_OK on success ESP_ERR_INVALID_ARG mac is NULL

ESP_ERR_INVALID_MAC CUSTOM_MAC address has not been set

ESP_ERR_INVALID_VERSION An invalid MAC version field was read from BLK3 of EFUSE (for esp32)

ESP_ERR_INVALID_CRC An invalid MAC CRC was read from BLK3 of EFUSE (for esp32)

esp_err_t esp_efuse_mac_get_default (uint8_t* mac)

Return base MAC address which is factory-programmed by Espressif in EFUSE.

参数 mac – base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4)

返回 ESP_OK on success ESP_ERR_INVALID_ARG mac is NULL

Espressif Systems 2003 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
# Chapter 2. API 参考

**esp_err_t esp_read_mac**(uint8_t *mac, esp_mac_type_t type)

Read base MAC address and set MAC address of the interface.

This function first get base MAC address using esp_base_mac_addr_get(). Then calculates the MAC address of the specific interface requested, refer to ESP-IDF Programming Guide for the algorithm.

The MAC address set by the esp_iface_mac_addr_set() function will not depend on the base MAC address.

参数
- **mac** – base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4)
- **type** – Type of MAC address to return

返回 ESP_OK on success

**esp_err_t esp_derive_local_mac**(uint8_t *local_mac, const uint8_t *universal_mac)

Derive local MAC address from universal MAC address.

This function copies a universal MAC address and then sets the “locally administered” bit (bit 0x2) in the first octet, creating a locally administered MAC address.

If the universal MAC address argument is already a locally administered MAC address, then the first octet is XORed with 0x4 in order to create a different locally administered MAC address.

参数
- **local_mac** – base MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for IEEE 802.15.4)
- **universal_mac** – Source universal MAC address, length: 6 bytes.

返回 ESP_OK on success

**esp_err_t esp_iface_mac_addr_set**(const uint8_t *mac, esp_mac_type_t type)

Set custom MAC address of the interface. This function allows you to overwrite the MAC addresses of the interfaces set by the base MAC address.

参数
- **mac** – MAC address, length: 6 bytes/8 bytes. length: 6 bytes for MAC-48 8 bytes for EUI-64(used for ESP_MAC_IEEE802154 type)
- **type** – Type of MAC address

返回 ESP_OK on success

**size_t esp_mac_addr_len_get**(esp_mac_type_t type)

Return the size of the MAC type in bytes.

If CONFIG_IEEE802154_ENABLED is set then for these types: ESP_MAC_IEEE802154, ESP_MAC_BASE, ESP_MAC_EFUSE_FACTORY and ESP_MAC_EFUSE_CUSTOM the MAC size is 8 bytes. If CONFIG_IEEE802154_ENABLED is not set then for all types it returns 6 bytes.

参数 **type** – Type of MAC address

返回 0 MAC type not found (not supported) 6 bytes for MAC-48. 8 bytes for EUI-64.

**Macros**

MAC2STR (a)

MACSTR

**Enumerations**

enum esp_mac_type_t

Values:

Espressif Systems  2004  Release v5.1-dev-2066-g7869f4e151
enumerator ESP_MAC_WIFI_STA
enumerator ESP_MAC_WIFI_SOFTAP
enumerator ESP_MAC_BT
enumerator ESP_MAC_ETH
enumerator ESP_MAC_IEEE802154
enumerator ESP_MAC_BASE
enumerator ESP_MAC_efuse_factory
enumerator ESP_MAC_efuse_custom

Header File
- components/esp_hwsupport/include/esp_chip_info.h

Functions

```c
void esp_chip_info(esp_chip_info_t *out_info)
```

Fill an `esp_chip_info_t` structure with information about the chip.

参数 out_info [out] structure to be filled

Structures

```c
struct esp_chip_info_t
```

The structure represents information about the chip.

Public Members

```c
esp_chip_model_t model
```

chip model, one of esp_chip_model_t

```c
uint32_t features
```

bit mask of CHIP_FEATURE_x feature flags

```c
uint16_t revision
```

chip revision number (in format MXX; where M - wafer major version, XX - wafer minor version)

```c
uint8_t cores
```

number of CPU cores
Macros

CHIP_FEATURE_EMB_FLASH
Chip has embedded flash memory.

CHIP_FEATURE_WIFI_BGN
Chip has 2.4GHz WiFi.

CHIP_FEATURE_BLE
Chip has Bluetooth LE.

CHIP_FEATURE_BT
Chip has Bluetooth Classic.

CHIP_FEATURE_IEEE802154
Chip has IEEE 802.15.4.

CHIP_FEATURE_EMB_PSRAM
Chip has embedded psram.

Enumerations

enum esp_chip_model_t
Chip models.

Values:

enumerator CHIP_ESP32
ESP32.

enumerator CHIP_ESP32S2
ESP32-S2.

enumerator CHIP_ESP32S3
ESP32-S3.

enumerator CHIP_ESP32C3
ESP32-C3.

enumerator CHIP_ESP32H4
ESP32-H4.

enumerator CHIP_ESP32C2
ESP32-C2.

enumerator CHIP_ESP32C6
ESP32-C6.

Header File

* components/esp_hw_support/include/esp_cpu.h
Chapter 2. API 参考

**Functions**

```c
void esp_cpu_stall (int core_id)

Stall a CPU core.

参数 core_id – The core’s ID
```

```c
void esp_cpu_unstall (int core_id)

Resume a previously stalled CPU core.

参数 core_id – The core’s ID
```

```c
void esp_cpu_reset (int core_id)

Reset a CPU core.

参数 core_id – The core’s ID
```

```c
void esp_cpu_wait_for_intr (void)

Wait for Interrupt.

This function causes the current CPU core to execute its Wait For Interrupt (WFI or equivalent) instruction. After executing this function, the CPU core will stop execution until an interrupt occurs.

int esp_cpu_get_core_id (void)

Get the current core’s ID.

This function will return the ID of the current CPU (i.e., the CPU that calls this function).

返回 The current core’s ID [0..SOC_CPU_CORES_NUM - 1]
```

```c
void *esp_cpu_get_sp (void)

Read the current stack pointer address.

返回 Stack pointer address
```

```c
esp_cpu_cycle_count_t esp_cpu_get_cycle_count (void)

Get the current CPU core’s cycle count.

Each CPU core maintains an internal counter (i.e., cycle count) that increments every CPU clock cycle.

返回 Current CPU’s cycle count, 0 if not supported.
```

```c
void esp_cpu_set_cycle_count (esp_cpu_cycle_count_t cycle_count)

Set the current CPU core’s cycle count.

Set the given value into the internal counter that increments every CPU clock cycle.

参数 cycle_count – CPU cycle count
```

```c
void *esp_cpu_pc_to_addr (uint32_t pc)

Convert a program counter (PC) value to address.

If the architecture does not store the true virtual address in the CPU’s PC or return addresses, this function will convert the PC value to a virtual address. Otherwise, the PC is just returned.

参数 pc – PC value

返回 Virtual address
```

```c
void esp_cpu_intr_get_desc (int core_id, int intr_num, esp_cpu_intr_desc_t *intr_desc_ret)

Get a CPU interrupt’s descriptor.

Each CPU interrupt has a descriptor describing the interrupt’s capabilities and restrictions. This function gets the descriptor of a particular interrupt on a particular CPU.

参数

- core_id – [in] The core’s ID
- intr_num – [in] Interrupt number
- intr_desc_ret – [out] The interrupt’s descriptor
```
void **esp_cpu_intr_set_ivt_addr** (const void *ivt_addr)
Set the base address of the current CPU’s Interrupt Vector Table (IVT)

- **Parameter ivt_addr**: Interrupt Vector Table’s base address

bool **esp_cpu_intr_has_handler** (int intr_num)
Check if a particular interrupt already has a handler function.

Check if a particular interrupt on the current CPU already has a handler function assigned.

**Note**: This function simply checks if the IVT of the current CPU already has a handler assigned.

- **Parameter intr_num**: Interrupt number (from 0 to 31)
- **Return**: True if the interrupt has a handler function, false otherwise.

void **esp_cpu_intr_set_handler** (int intr_num, **esp_cpu_intr_handler_t** handler, void *handler_arg)
Set the handler function of a particular interrupt.

Assign a handler function (i.e., ISR) to a particular interrupt on the current CPU.

**Note**: This function simply sets the handler function (in the IVT) and does not actually enable the interrupt.

- **Parameters**
  - **intr_num**: Interrupt number (from 0 to 31)
  - **handler**: Handler function
  - **handler_arg**: Argument passed to the handler function

void **esp_cpu_intr_get_handler_arg** (int intr_num)
Get a handler function’s argument of.

Get the argument of a previously assigned handler function on the current CPU.

- **Parameter intr_num**: Interrupt number (from 0 to 31)
- **Return**: The argument passed to the handler function

void **esp_cpu_intr_enable** (uint32_t intr_mask)
Enable particular interrupts on the current CPU.

- **Parameter intr_mask**: Bit mask of the interrupts to enable

void **esp_cpu_intr_disable** (uint32_t intr_mask)
Disable particular interrupts on the current CPU.

- **Parameter intr_mask**: Bit mask of the interrupts to disable

**uint32_t** **esp_cpu_intr_get_enabled_mask** (void)
Get the enabled interrupts on the current CPU.

- **Return**: Bit mask of the enabled interrupts

void **esp_cpu_intr_edge_ack** (int intr_num)
Acknowledge an edge interrupt.

- **Parameter intr_num**: Interrupt number (from 0 to 31)

void **esp_cpu_configure_region_protection** (void)
Configure the CPU to disable access to invalid memory regions.
**esp_err_t esp_cpu_set_breakpoint** (int bp_num, const void *bp_addr)

Set and enable a hardware breakpoint on the current CPU.

**备注**: This function is meant to be called by the panic handler to set a breakpoint for an attached debugger during a panic.

**备注**: Overwrites previously set breakpoint with same breakpoint number.

**参数**
- **bp_num** – Hardware breakpoint number [0..SOC_CPU_BREAKPOINTS_NUM - 1]
- **bp_addr** – Address to set a breakpoint on

**返回** ESP_OK if breakpoint is set. Failure otherwise

**esp_err_t esp_cpu_clear_breakpoint** (int bp_num)

Clear a hardware breakpoint on the current CPU.

**备注**: Clears a breakpoint regardless of whether it was previously set

**参数** **bp_num** – Hardware breakpoint number [0..SOC_CPU_BREAKPOINTS_NUM - 1]

**返回** ESP_OK if breakpoint is cleared. Failure otherwise

**esp_err_t esp_cpu_set_watchpoint** (int wp_num, const void *wp_addr, size_t size, esp_cpu_watchpoint_trigger_t trigger)

Set and enable a hardware watchpoint on the current CPU.

Set and enable a hardware watchpoint on the current CPU, specifying the memory range and trigger operation. Watchpoints will break/panic the CPU when the CPU accesses (according to the trigger type) on a certain memory range.

**备注**: Overwrites previously set watchpoint with same watchpoint number.

**参数**
- **wp_num** – Hardware watchpoint number [0..SOC_CPU_WATCHPOINTS_NUM - 1]
- **wp_addr** – Watchpoint’s base address
- **size** – Size of the region to watch. Must be one of 2^n, with n in [0..6].
- **trigger** – Trigger type

**返回** ESP_ERR_INVALID_ARG on invalid arg, ESP_OK otherwise

**esp_err_t esp_cpu_clear_watchpoint** (int wp_num)

Clear a hardware watchpoint on the current CPU.

**备注**: Clears a watchpoint regardless of whether it was previously set

**参数** **wp_num** – Hardware watchpoint number [0..SOC_CPU_WATCHPOINTS_NUM - 1]

**返回** ESP_OK if watchpoint was cleared. Failure otherwise.

**bool esp_cpu_dbgr_is_attached**(void)

Check if the current CPU has a debugger attached.

**返回** True if debugger is attached, false otherwise
Chapter 2. API 參考

void esp_cpu_dbgbr_break (void)
    Trigger a call to the current CPU’s attached debugger.

intptr_t esp_cpu_get_call_addr (intptr_t return_address)
    Given the return address, calculate the address of the preceding call instruction. This is typically used to answer the question “where was the function called from?”.
    参数 return_address – The value of the return address register. Typically set to the value of __builtin_return_address(0).
    返回 Address of the call instruction preceding the return address.

bool esp_cpu_compare_and_set (volatile uint32_t *addr, uint32_t compare_value, uint32_t new_value)
    Atomic compare-and-set operation.
    参数
        • addr – Address of atomic variable
        • compare_value – Value to compare the atomic variable to
        • new_value – New value to set the atomic variable to
    返回 Whether the atomic variable was set or not

Structures

struct esp_cpu_intr_desc_t
    CPU interrupt descriptor.
    Each particular CPU interrupt has an associated descriptor describing that particular interrupt’s characteristics. Call esp_cpu_intr_get_desc() to get the descriptors of a particular interrupt.

Public Members

int priority
    Priority of the interrupt if it has a fixed priority, (-1) if the priority is configurable.

esp_cpu_intr_type_t type
    Whether the interrupt is an edge or level type interrupt, ESP_CPU_INTR_TYPE_NA if the type is configurable.

uint32_t flags
    Flags indicating extra details.

Macros

ESP_CPU_INTR_DESC_FLAG_SPECIAL
    Interrupt descriptor flags of esp_cpu_intr_desc_t.
    The interrupt is a special interrupt (e.g., a CPU timer interrupt)

ESP_CPU_INTR_DESC_FLAG_RESVD
    The interrupt is reserved for internal use

Type Definitions

typedef uint32_t esp_cpu_cycle_count_t
    CPU cycle count type.
    This data type represents the CPU’s clock cycle count
typedef void (*esp_cpu_intr_handler_t)(void *arg)
  CPU interrupt handler type.

Enumerations

enum esp_cpu_intr_type_t
  CPU interrupt type.

Values:

enumerator ESP_CPU_INTR_TYPE_LEVEL
enumerator ESP_CPU_INTR_TYPE_EDGE
enumerator ESP_CPU_INTR_TYPE_NA

enum esp_cpu_watchpoint_trigger_t
  CPU watchpoint trigger type.

Values:

enumerator ESP_CPU_WATCHPOINT_LOAD
enumerator ESP_CPU_WATCHPOINT_STORE
enumerator ESP_CPU_WATCHPOINT_ACCESS

Header File

  - components/esp_app_format/include/esp_app_desc.h

Functions

const esp_app_desc_t *esp_app_get_description (void)
  Return esp_app_desc structure. This structure includes app version.
  Return description for running app.

int esp_app_get_elf_sha256 (char *dst, size_t size)
  Fill the provided buffer with SHA256 of the ELF file, formatted as hexadecimal, null-terminated. If the buffer size is not sufficient to fit the entire SHA256 in hex plus a null terminator, the largest possible number of bytes will be written followed by a null.

  参数
  - dst - Destination buffer
  - size - Size of the buffer

  返回 Number of bytes written to dst (including null terminator)

Structures

struct esp_app_desc_t
  Description about application.
Public Members

- `uint32_t magic_word`
  Magic word ESP_APP_DESC_MAGIC_WORD

- `uint32_t secure_version`
  Secure version

- `uint32_t reserv1[2]`
  reserv1

- `char version[32]`
  Application version

- `char project_name[32]`
  Project name

- `char time[16]`
  Compile time

- `char date[16]`
  Compile date

- `char idf_ver[32]`
  Version IDF

- `uint8_t app_elf_sha256[32]`
  sha256 of elf file

- `uint32_t reserv2[20]`
  reserv2

Macros

- `ESP_APP_DESC_MAGIC_WORD`
  The magic word for the esp_app_desc structure that is in DROM.

2.10.20  空中升级 (OTA)

OTA 流程概览

OTA 升级机制可以让设备在固件正常运行时根据接收数据（如通过 Wi-Fi 或蓝牙）进行自我更新。

要运行 OTA 机制，需配置设备的分区表，该分区表至少包括两个 OTA 应用程序分区（即 `ota_0` 和 `ota_1`) 和一个 OTA 数据分区。

OTA 功能启动后，向当前未用于启动的 OTA 应用分区写入新的应用固件镜像。镜像验证后，OTA 数据分区更新，指定在下一次启动时使用该镜像。
OTA 数据分区

所有使用 OTA 功能的项目，其分区表必须包含一个 OTA 数据分区（类型为 data，子类型为 ota）。

工厂启动设置下，OTA 数据分区中应没有数据（所有字节擦写成 0xFF）。如果分区表中没有工厂应用程序，ESP-IDF 软件引导加载程序会启动工厂应用程序。如果分区表中没有工厂应用程序，则启动第一个可用的 OTA 分区（通常是 ota_0）。

第一次 OTA 升级后，OTA 数据分区更新。指定下一次启动哪个 OTA 应用程序分区。

OTA 数据分区是两个 0x2000 字节大小的 flash 虚拟分区，防止写入时电源故障引发问题。两个扇区单独擦除、写入匹配数据，若存在不一致，则用计算器字段判定哪个扇区为最新数据。

应用程序回滚

应用程序回滚的主要目的是确保设备在更新后正常工作。如果新版应用程序出现严重错误，该功能可使设备回滚到之前正常运行的应用版本。在使能回滚并且 OTA 升级应用程序至新版本后，可能出现的结果如下：

- 应用程序运行正常，esp_ota_mark_app_valid_cancel_rollback() 将正在运行的程序状态标记为 ESP_OTA_IMG_VALID，启动此应用程序无限制。
- 应用程序出现严重错误，无法继续工作，必须回滚到之前的版本，esp_ota_mark_app_invalid_rollback_and_reboot() 将正在运行的版本标记为 ESP_OTA_IMG_INVALID 然后复位。引导加载程序不执行此版本，而是启动之前正常运行的版本。
- 如果 CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 使能，则无需调用函数便可复位，回滚至之前的任意版本。

注解：应用程序的状态不是写到程序的二进制镜像，而是写入 otadata 分区。该分区有一个 ota_seq 计数器，该计数器是 OTA 应用分区的指针，指向下次启动时启动应用所占的分区（ota_0, ota_1, …）。在 otadata 分区中。

应用程序 OTA 状态

状态控制了选取启动应用程序的过程：

<table>
<thead>
<tr>
<th>状态</th>
<th>引导加载程序选取启动应用程序的限制</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_OTA_IMG_VALID</td>
<td>可以选取。</td>
</tr>
<tr>
<td>ESP_OTA_IMG_INVALID</td>
<td>无限制，可以选取。</td>
</tr>
<tr>
<td>ESP_OTA_IMG_ABORTED</td>
<td>无限制，可以选取。</td>
</tr>
<tr>
<td>ESP_OTA_IMG_NEW</td>
<td>可选取。</td>
</tr>
<tr>
<td>ESP_OTA_IMG_PENDING_VERIFY</td>
<td>可选取。</td>
</tr>
<tr>
<td>ESP_OTA_IMG_PENDING_VERIFY</td>
<td>可选取。</td>
</tr>
<tr>
<td>CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE</td>
<td>会选取一次。在开启过程中，状态立变位ESP_OTA_IMG_PENDING_VERIFY。</td>
</tr>
<tr>
<td>ESP_OTA_IMG_ABORTED</td>
<td>会选取，状态变为 “ESP_OTA_IMG_ABORTED”。</td>
</tr>
</tbody>
</table>

如 果 CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 没有使能（默认情况），则 esp_ota_mark_app_valid_cancel_rollback() 和 esp_ota_mark_app_invalid_rollback_and_reboot() 为可选功能，ESP_OTA_IMG_NEW 和 ESP_OTA_IMG_PENDING_VERIFY 不会使用。

Kconfig 中的 CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 可以帮助用户追踪新版应用程序的第一次启动。应用程序需调用 esp_ota_mark_app_valid_cancel_rollback() 函数确认可以运行，否则将在重启时回滚至旧版本。该功能可让用户在启动阶段控制应用程序的可操作性。新版应用程序仅有一次机会尝试是否能成功启动。

回滚过程

CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 使能时，回滚过程如下：

- 新版应用程序下载成功，esp_ota_set_boot_partition() 函数将分区设为可启动，状态设为 ESP_OTA_IMG_NEW。该状态表示应用程序为新版本，第一次启动需要检测。
- 重新启动 esp_restart()。
• 引导加载程序检查 ESP OTA_IMG_PENDING_VERIFY 状态，如有设置，则将其写入 ESP OTA_IMG_ABORTED.

• 引导加载程序选取一个新的应用程序来引导，这样应用程序状态就不再设置为 ESP OTA_IMG_INVALID 或 ESP OTA_IMG_ABORTED.

• 引导加载程序检查所选取的新版应用程序，若状态设置为 ESP OTA_IMG_NEW，则写入 ESP OTA_IMG_PENDING_VERIFY. 该状态表示，需确认应用程序的可操作性，如不确认，发生重启，则状态会重写 ESP OTA_IMG_ABORTED（见上文），该应用程序不可再启动，将回滚至上一版本。

• 新版应用程序启动时，应进行自测。

• 若通过自测，则必须调用函数 esp_ota_mark_app_valid_cancel_rollback()，为新版应用程序等待确认其可操作性（ESP OTA_IMG_PENDING_VERIFY 状态）。

• 若未通过自测，则调用函数 esp_ota_mark_app_invalid_rollack_and_reboot()，回滚至之前能正常工作的应用程序版本，同时将无效的新版本应用程序设置为 ESP OTA_IMG_INVALID。

• 如果新版应用程序可操作性没有确认，则状态一直为 ESP OTA_IMG_PENDING_VERIFY. 下一次启动时，状态变更为 ESP OTA_IMG_ABORTED, 阻止其再次启动，之后回滚到之前的版本。

意外复位 如果在新版本应用第一次启动时发生断电或意外崩溃，则会回滚至之前正常运行的版本。

建议：尽快完成自测，防止因断电回滚。

只有 OTA 分区可以回滚。工厂分区不会回滚。

启动无效/中止的应用程序 用户可以启动此前设置为 ESP OTA_IMG_INVALID 或 ESP OTA_IMG_ABORTED 的应用程序

- 获取最后一个无效分区 esp_ota_get_last_invalid_partition()。
- 将获取的分区传递给 esp_ota_set_boot_partition()，更新 otadata。
- 重启 esp_restart()。引导加载程序会启动指定应用程序。

要确定是否在应用程序启动时进行自测，可以调用 esp_ota_get_state_partition() 函数。如果结果为 ESP OTA_IMG_PENDING_VERIFY，则需要自测，后续确认应用程序的可操作性。

如何设置状态 下文简单描述了如何设置应用程序状态：

- ESP OTA_IMG_INVALID 由函数 esp_ota_mark_app_valid_cancel_rollack() 设置。

- 如果 CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 没有使能，ESP OTA_IMAGE_UNDEFFINED 由函数 esp_ota_set_boot_partition() 设置。

- 如果 CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 没有使能，ESP OTA_IMG_NEW 由函数 esp_ota_set_boot_partition() 设置。

- ESP OTA_IMG_INVALID 由函数 esp_ota_mark_app_invalid_rollack_and_reboot() 设置。

- 如果应用程序的可操作性无法确认，发生重启（CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 使能），则设置 ESP OTA_IMG_ABORTED。

- 如果 CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE 使能，选取的应用程序状态为 ESP OTA_IMG_NEW, 则在引导加载程序中设置 ESP OTA_IMG_PENDING_VERIFY。

防御回滚

防御回滚机制可以防止回滚到安全版本号低于芯片 eFuse 中烧录程序的应用程序版本。

设置 CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK, 启动防御回滚机制。在引导加载程序中选取可启动的应用程序，会额外检查芯片和应用程序镜像的安全版本号。可启动固件中的应用安全版本号必须等于或高于芯片中的应用安全版本号。

CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK 和 CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK_ENABLE 一起使用。此时，只有安全版本号等于或高于芯片中的应用安全版本号时才会回滚。

Epressif Systems 2014 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
典型的防回滚机制

- 新发布的固件解决了此前版本的安全问题。
- 开发者在确保固件可以运行之后，增加安全版本号，发布固件。
- 下载新版应用程序。
- 运行函数 `esp_ota_set_boot_partition()`，将新版应用程序设为可启动。如果新版应用程序的安全版本号低于芯片中的应用安全版本号，新版应用程序会被擦除，无法更新到新固件。
- 重新启动。
- 在引导加载程序中选取安全版本号等于或高于芯片中应用安全版本号的应用程序。如果 `otadata` 处于初始阶段，通过串行通道加载了安全版本号高于芯片中应用安全版本号的固件，则引导加载程序中 `eFuse` 的安全版本号会立即更新。
- 新版应用程序启动，之后进行可操作性检测，如果通过检测，则调用函数 `esp_ota_mark_app_valid_cancel_rollback()`，将应用程序标记为 `ESP_OOTA_IMG_VALID`，更新芯片中应用程序的安全版本号。注意，如果调用函数 `esp_ota_mark_app_invalid_rollback_and_reboot()`，可能会因为设备中没有可启动的应用程序而回滚失败，返回 ESP_ERR_OTA.Rollback_FAILED 错误。应用程序状态一直为 `ESP_OOTA_IMG_PENDING_VERIFY`。
- 如果运行的应用程序处于 `ESP_OOTA_IMG_VALID` 状态，则可再次更新。

建议：

如果想避免因服务器应用程序的安全版本号低于运行的应用程序，造成不必要的下载和擦除，必须从镜像的第一个包中获取 `new_app_info.secure_version`，和 `eFuse` 的安全版本号比较。如果 `esp_efuse_check_secure_version(new_app_info.secure_version)` 函数为真，则下载继续，反之则中断。

```c
bool image_header_was_checked = false;
while (1) {
   int data_read = esp_http_client_read(client, ota_write_data, BUFFSIZE);
   ...
   if (data_read > 0) {
      if (image_header_was_checked == false) {
         esp_app_desc_t new_app_info;
         if (data_read > sizeof(esp_image_header_t) + sizeof(esp_image_segment_header_t) + sizeof(esp_app_desc_t)) {
            // check current version with downloading
            if (esp_efuse_check_secure_version(new_app_info.secure_version) == false) {
               ESP_LOGE(TAG, "This a new app can not be downloaded due to a secure version is lower than stored in efuse.");
               http_cleanup(client);
               task_fatal_error();
            }
         }
         image_header_was_checked = true;
         esp_ota_begin(update_partition, OTA_SIZE_UNKNOWN, &update_handle);
         }
      esp_ota_write(update_handle, (const void *)ota_write_data, data_read);
   }
   ...
```

限制：

- `secure_version` 字段最多有 32 位。也就是说，防回滚最多可以做 32 次。用户可以使用 `CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD` 减少该 `eFuse` 字段的长度。
- 防回滚仅在 `eFuse` 编码机制设置为 `NONE` 时生效。
- 防回滚不支持工厂和测试分区，因此分区表中不应有设置为 `工厂` 或 `测试` 的分区。
security_version:

- 存储在应用程序镜像中的 esp_app_desc 里。版本号用 CONFIG_BOOTLOADER_APP_SECURE_VERSION 设置。

- ESP32 中版本号存储在 eFuse 的 EFUSE_BLK3_RDATA4_REG 里（若 eFuse 的位烧号为 1，则永远无法恢复为 0）。寄存器设置了多少位，应用程序的安全版本号就为多少。

没有安全启动的安全 OTA 升级

即便硬件安全启动没有使能，也可验证已签名的 OTA 升级。可通过设置 CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT 和 CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT 实现。

具体可参考 Signed App Verification Without Hardware Secure Boot。

OTA 工具 (otatool.py)

app_update 组件中有 otatool.py 工具，用于在目标设备上完成下列 OTA 区分相关操作：

- 读取 otadata 分区 (read_otadata)
- 擦除 otadata 分区，将设备复位至工厂应用程序 (erase_otadata)
- 切换 OTA 分区 (switch_ota_partition)
- 擦除 OTA 分区 (erase_ota_partition)
- 写入 OTA 分区 (write_ota_partition)
- 读取 OTA 分区 (read_ota_partition)

用户若想通过编程方式完成相关操作，可从另一个 Python 脚本导入并使用该 OTA 工具，或者从 Shell 脚本调用该 OTA 工具。前者可使用工具的 Python API，后者可使用命令行界面。

Python API 首先，确保已导入 otatool 模块。

```python
import sys
import os

idf_path = os.environ['IDF_PATH']  # 从环境中获取 IDF_PATH 的值
otatool_dir = os.path.join(idf_path, 'components', 'app_update')  # otatool.py 位于 $IDF_PATH/components/app_update 下

sys.path.append(otatool_dir)  # 使 Python 寻找 otatool 模块
from otatool import *  # 导入 otatool 模块内的所有名称
```

要使用 OTA 工具的 Python API，第一步是创建 OtoolTarget 对象：

```python
# 创建 partool.py 的目标设备，并将目标设备连接到串行端口 /dev/ttyUSB1
target = OtoolTarget('/dev/ttyUSB1')
```

现在，可使用创建的 OtoolTarget 在目标设备上完成操作：

```python
# 擦除 otadata，将设备复位至工厂应用程序
target.erase_otadata()

# 擦除 OTA 应用程序分区 0
target.erase_ota_partition(0)

# 将启动分区切换至 OTA 应用程序分区 1
target.switch_ota_partition(1)

# 读取 OTA 分区 ‘ota_3’，将内容保存至文件 ‘ota_3.bin’
target.read_ota_partition("ota_3", "ota_3.bin")
```
要操作的 OTA 分区通过应用程序分区序号或分区名称指定。
更多关于 Python API 的信息，请查看 OTA 工具的代码注释。

**命令行界面**  `otatool.py` 的命令行界面具有如下结构：

```
otatool.py [command-args] [subcommand] [subcommand-args]

- command-args - 执行主命令（`otatool.py`）所需的实际参数，多与目标设备有关
- subcommand - 要执行的操作
- subcommand-args - 所选操作的实际参数
```

更多信息可用 `-help` 指令查看：

```
# 撤除 otadata，将设备复位至工厂应用程序
otatool.py --port "/dev/ttyUSB1" erase_otadata

# 撤除 OTA 应用程序分区 0
otatool.py --port "/dev/ttyUSB1" erase_ota_partition --slot 0

# 将启动分区切换至 OTA 应用程序分区 1
otatool.py --port "/dev/ttyUSB1" switch_ota_partition --slot 1

# 读取 OTA 分区 'ota_3'，将内容保存至文件 'ota_3.bin'
otatool.py --port "/dev/ttyUSB1" read_ota_partition --name=ota_3 --output=ota_3.bin
```

**相关文档**

- 分区表
- `SPI Flash` 和分区 API
- `ESP HTTPS OTA`

**应用程序示例**

端对端的 OTA 固件升级示例请参考 system/ota。

**API 参考**

**Header File**

```
components/app_update/include/esp_ota_ops.h
```

**Functions**

```
const esp_app_desc_t *esp_ota_get_app_description (void)
```

```
Return esp_app_desc structure. This structure includes app version.
```

```
Return description for running app.
```
This API is present for backward compatibility reasons. Alternative function with the same functionality is `esp_app_get_description`.

**Return** Pointer to esp_app_desc structure.

```c
int esp_ota_get_app_elf_sha256 (char *dst, size_t size)
```

Fill the provided buffer with SHA256 of the ELF file, formatted as hexadecimal, null-terminated. If the buffer size is not sufficient to fit the entire SHA256 in hex plus a null terminator, the largest possible number of bytes will be written followed by a null.

This API is present for backward compatibility reasons. Alternative function with the same functionality is `esp_app_get_elf_sha256`.

**Parameters**
- `dst` – Destination buffer
- `size` – Size of the buffer

**Return** Number of bytes written to dst (including null terminator)

```c
esp_err_t esp_ota_begin (const esp_partition_t *partition, size_t image_size, esp_ota_handle_t *out_handle)
```

Commence an OTA update writing to the specified partition.

The specified partition is erased to the specified image size.

If image size is not yet known, pass OTA_SIZE_UNKNOWN which will cause the entire partition to be erased.

On success, this function allocates memory that remains in use until esp_ota_end() is called with the returned handle.

**Note:** If the rollback option is enabled and the running application has the ESP_OTA_IMG_PENDING_VERIFY state then it will lead to the ESP_ERR_OTA_ROLLBACK_INVALID_STATE error. Confirm the running app before to run download a new app, use esp_ota_mark_app_valid_cancel_rollback() function for it (this should be done as early as possible when you first download a new application).

**Parameters**
- `partition` – Pointer to info for partition which will receive the OTA update. Required.
- `image_size` – Size of new OTA app image. Partition will be erased in order to receive this size of image. If 0 or OTA_SIZE_UNKNOWN, the entire partition is erased.
- `out_handle` – On success, returns a handle which should be used for subsequent esp_ota_write() and esp_ota_end() calls.

**Return**
- ESP_OK: OTA operation commenced successfully.
- ESP_ERR_INVALID_ARG: partition or out_handle arguments were NULL, or partition doesn’t point to an OTA app partition.
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_OTA_PARTITION_CONFLICT: Partition holds the currently running firmware, cannot update in place.
- ESP_ERR_NOT_FOUND: Partition argument not found in partition table.
- ESP_ERR_OTA_SELECT_INFO_INVALID: The OTA data partition contains invalid data.
- ESP_ERR_OTA_INVALID_SIZE: Partition doesn’t fit in configured flash size.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_ROLLBACK_INVALID_STATE: If the running app has not confirmed state. Before performing an update, the application must be valid.
**esp_err_t esp_ota_write** *(esp_ota_handle_t handle, const void *data, size_t size)*

Write OTA update data to partition.

This function can be called multiple times as data is received during the OTA operation. Data is written sequentially to the partition.

**参数**
- **handle** - Handle obtained from esp_ota_begin
- **data** - Data buffer to write
- **size** - Size of data buffer in bytes.

**返回**
- ESP_OK: Data was written to flash successfully.
- ESP_ERR_INVALID_ARG: handle is invalid.
- ESP_ERR_OTA_VALIDATE_FAILED: First byte of image contains invalid app image magic byte.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_SELECT_INFO_INVALID: OTA data partition has invalid contents

**esp_err_t esp_ota_write_with_offset** *(esp_ota_handle_t handle, const void *data, size_t size, uint32_t offset)*

Write OTA update data to partition at an offset.

This function can write data in non-contiguous manner. If flash encryption is enabled, data should be 16 bytes aligned.

**参数**
- **handle** - Handle obtained from esp_ota_begin
- **data** - Data buffer to write
- **size** - Size of data buffer in bytes
- **offset** - Offset in flash partition

**返回**
- ESP_OK: Data was written to flash successfully.
- ESP_ERR_INVALID_ARG: handle is invalid.
- ESP_ERR_OTA_VALIDATE_FAILED: First byte of image contains invalid app image magic byte.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_SELECT_INFO_INVALID: OTA data partition has invalid contents

**esp_err_t esp_ota_end** *(esp_ota_handle_t handle)*

Finish OTA update and validate newly written app image.

**备注:** After calling esp_ota_end(), the handle is no longer valid and any memory associated with it is freed (regardless of result).
Chapter 2. API

- ESP_ERR_OTA_VALIDATE_FAILED: OTA image is invalid (either not a valid app image, or - if secure boot is enabled - signature failed to verify.)
- ESP_ERR_INVALID_STATE: If flash encryption is enabled, this result indicates an internal error writing the final encrypted bytes to flash.

\[ \text{esp_err_t} \ \text{esp_ota_abort} (\text{esp_ota_handle_t} \ handle) \]
Abort OTA update, free the handle and memory associated with it.

\begin{itemize}
  \item \text{handle} - obtained from esp_ota_begin().
\end{itemize}

\begin{itemize}
  \item ESP_OK: Handle and its associated memory is freed successfully.
  \item ESP_ERR_NOT_FOUND: OTA handle was not found.
\end{itemize}

\[ \text{esp_err_t} \ \text{esp_ota_set_boot_partition} (\text{const} \ \text{esp_partition_t} *\text{partition}) \]
Configure OTA data for a new boot partition.

备注: If this function returns ESP_OK, calling esp_restart() will boot the newly configured app partition.

\begin{itemize}
  \item \text{partition} - Pointer to info for partition containing app image to boot.
\end{itemize}

\begin{itemize}
  \item ESP_OK: OTA data updated, next reboot will use specified partition.
  \item ESP_ERR_INVALID_ARG: partition argument was NULL or didn’t point to a valid OTA partition of type “app”.
  \item ESP_ERR_OTA_VALIDATE_FAILED: Partition contained invalid app image. Also returned if secure boot is enabled and signature validation failed.
  \item ESP_ERR_NOT_FOUND: OTA data partition not found.
  \item ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash erase or write failed.
\end{itemize}

\[ \text{const} \ \text{esp_partition_t} *\text{esp_ota_get_boot_partition} (\text{void}) \]
Get partition info of currently configured boot app.

- If esp_ota_set_boot_partition() has been called, the partition which was set by that function will be returned.
- If esp_ota_set_boot_partition() has not been called, the result is usually the same as esp_ota_get_running_partition(). The two results are not equal if the configured boot partition does not contain a valid app (meaning that the running partition will be an app that the bootloader chose via fallback).

If the OTA data partition is not present or not valid then the result is the first app partition found in the partition table. In priority order, this means: the factory app, the first OTA app slot, or the test app partition.

Note that there is no guarantee the returned partition is a valid app. Use esp_image_verify(ESP_IMAGE_VERIFY, ...) to verify if the returned partition contains a bootable image.

\begin{itemize}
  \item \text{return} Pointer to info for partition structure, or NULL if partition table is invalid or a flash read operation failed. Any returned pointer is valid for the lifetime of the application.
\end{itemize}

\[ \text{const} \ \text{esp_partition_t} *\text{esp_ota_get_running_partition} (\text{void}) \]
Get partition info of currently running app.

This function is different to esp_ota_get_boot_partition() in that it ignores any change of selected boot partition caused by esp_ota_set_boot_partition(). Only the app whose code is currently running will have its partition information returned.

The partition returned by this function may also differ from esp_ota_get_boot_partition() if the configured boot partition is somehow invalid, and the bootloader fell back to a different app partition at boot.

\begin{itemize}
  \item \text{return} Pointer to info for partition structure, or NULL if no partition is found or flash read operation failed. Returned pointer is valid for the lifetime of the application.
\end{itemize}
const esp_partition_t *esp_ota_get_next_update_partition (const esp_partition_t *start_from)

Return the next OTA app partition which should be written with a new firmware.

Call this function to find an OTA app partition which can be passed to esp_ota_begin().

Finds next partition round-robin, starting from the current running partition.

参数 start_from — If set, treat this partition info as describing the current running partition. Can be NULL, in which case esp_ota_get_running_partition() is used to find the currently running partition. The result of this function is never the same as this argument.

参数 start_from — If set, treat this partition info as describing the current running partition. Can be NULL, in which case esp_ota_get_running_partition() is used to find the currently running partition. The result of this function is never the same as this argument.

esp_err_t esp_ota_get_partition_description (const esp_partition_t *partition, esp_app_desc_t *app_desc)

Returns esp_app_desc structure for app partition. This structure includes app version.

参数

- partition — [in] Pointer to app partition. (only app partition)
- app_desc — [out] Structure of info about app.

返回

- ESP_OK Successful.
- ESP_ERR_NOT_FOUND app_desc structure is not found. Magic word is incorrect.
- ESP_ERR_NOT_SUPPORTED Partition is not application.
- ESP_ERR_INVALID_ARG Arguments is NULL or if partition’s offset exceeds partition size.
- ESP_ERR_INVALID_SIZE Read would go out of bounds of the partition.
- or one of error codes from lower-level flash driver.

uint8_t esp_ota_get_app_partition_count (void)

Returns number of OTA partitions provided in partition table.

参数

- Number of OTA partitions

esp_err_t esp_ota_mark_app_valid_cancel.Rollback (void)

This function is called to indicate that the running app is working well.

返回

- ESP_OK: if successful.

esp_err_t esp_ota_mark_app_invalid.Rollback_and_reboot (void)

This function is called to roll back to the previously workable app with reboot.

If rollback is successful then device will reset else API will return with error code. Checks applications on a flash drive that can be booted in case of rollback. If the flash does not have at least one app (except the running app) then rollback is not possible.

返回

- ESP_FAIL: if not successful.
- ESP_ERR_OTA.Rollback_FAILED: The rollback is not possible due to flash does not have any apps.

const esp_partition_t *esp_ota_get_last_invalid_partition (void)

Returns last partition with invalid state (ESP_OTA_IMG_INVALID or ESP_OTA_IMG_ABORTED).

参数

partition.
Chapter 2. API

• **partition** [in] Pointer to partition.
• **ota_state** [out] state of partition (if this partition has a record in odata).

返回
• ESP_OK: Successful.
• ESP_ERR_INVALID_ARG: partition or ota_state arguments were NULL.
• ESP_ERR_NOT_SUPPORTED: partition is not ota.
• ESP_ERR_NOT_FOUND: Partition table does not have odata or state was not found for given partition.

```c
esp_err_t esp_ota_erase_last_boot_app_partition (void)
```
Erase previous boot app partition and corresponding odata select for this partition.
When current app is marked to as valid then you can erase previous app partition.

返回
• ESP_OK: Successful, otherwise ESP_ERR.

```c
bool esp_ota_check_rollback_is_possible (void)
```
Checks applications on the slots which can be booted in case of rollback.
These applications should be valid (marked in odata as not UNDEFINED, INVALID or ABORTED and crc is good) and be able booted, and secure_version of app >= secure_version of efuse (if anti-rollback is enabled).

返回
• True: Returns true if the slots have at least one app (except the running app).
• False: The rollback is not possible.

** Macros **

**OTA_SIZE_UNKNOWN**
Used for esp_ota_begin() if new image size is unknown

**OTA_WITH_SEQUENTIAL_WRITES**
Used for esp_ota_begin() if new image size is unknown and erase can be done in incremental manner (assuming write operation is in continuous sequence)

**ESP_ERR_OTA_BASE**
Base error code for ota_ops api

**ESP_ERR_OTA_PARTITION_CONFLICT**
Error if request was to write or erase the current running partition

**ESP_ERR_OTA_SELECT_INFO_INVALID**
Error if OTA data partition contains invalid content

**ESP_ERR_OTA_VALIDATE_FAILED**
Error if OTA app image is invalid

**ESP_ERR_OTA_SMALL_SEC_VER**
Error if the firmware has a secure version less than the running firmware.

**ESP_ERR_OTA_ROLLBACK_FAILED**
Error if flash does not have valid firmware in passive partition and hence rollback is not possible

**ESP_ERR_OTA_ROLLBACK_INVALID_STATE**
Chapter 2. API 参考

Error if current active firmware is still marked in pending validation state (ESP_OOTA_IMG_PENDING_VERIFY), essentially first boot of firmware image post upgrade and hence firmware upgrade is not possible

Type Definitions

typedef uint32_t esp_ota_handle_t

Opaque handle for an application OTA update.

esp_ota_begin() returns a handle which is then used for subsequent calls to esp_ota_write() and esp_ota_end().

OTA 升级失败排查

图 38: OTA 升级失败时如何排查（点击放大）

2.10.21  Performance Monitor

The Performance Monitor component provides APIs to use ESP32 internal performance counters to profile functions and applications.

Espressif Systems 2023 Release v5.1-dev-2066-g7869f4e151 Submit Document Feedback
Application Example

An example which combines performance monitor is provided in `examples/system/perfmon` directory. This example initializes the performance monitor structure and execute them with printing the statistics.

High level API Reference

Header Files

- `perfmon/include/perfmon.h`

API Reference

Header File

- `components/perfmon/include/xtensa_perfmon_access.h`

Functions

```c
esp_err_t xtensa_perfmon_init(int id, uint16_t select, uint16_t mask, int kernelcnt, int tracelevel)
```

Init Performance Monitor.

Initialize performance monitor register with define values

<table>
<thead>
<tr>
<th>参数</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong> [in]</td>
<td>performance counter number</td>
</tr>
<tr>
<td><strong>select</strong> [in]</td>
<td>select value from PMCTRLx register</td>
</tr>
<tr>
<td><strong>mask</strong> [in]</td>
<td>mask value from PMCTRLx register</td>
</tr>
<tr>
<td><strong>kernelcnt</strong> [in]</td>
<td>kernelcnt value from PMCTRLx register</td>
</tr>
<tr>
<td><strong>tracelevel</strong> [in]</td>
<td>tracelevel value from PMCTRLx register</td>
</tr>
</tbody>
</table>

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if one of the arguments is not correct

```c
esp_err_t xtensa_perfmon_reset(int id)
```

Reset PM counter.

Reset PM counter. Writes 0 to the PMx register.

<table>
<thead>
<tr>
<th>参数</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong> [in]</td>
<td>performance counter number</td>
</tr>
</tbody>
</table>

返回

- ESP_OK on success
- ESP_ERR_INVALID_ARG if id out of range

```c
void xtensa_perfmon_start(void)
```

Start PM counters.

Start all PM counters synchronously. Write 1 to the PGM register

```c
void xtensa_perfmon_stop(void)
```

Stop PM counters.

Stop all PM counters synchronously. Write 0 to the PGM register

```c
uint32_t xtensa_perfmon_value(int id)
```

Read PM counter.

Read value of defined PM counter.

<table>
<thead>
<tr>
<th>参数</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong> [in]</td>
<td>performance counter number</td>
</tr>
</tbody>
</table>

返回

- Performance counter value
**esp_err_t xtensa_perfmon_overflow**(int id)

Read PM overflow state.

- **id** -[in] performance counter number

  - ESP_OK if there is no overflow (overflow = 0)
  - ESP_FAIL if overflow occur (overflow = 1)

**void xtensa_perfmon_dump**(void)

Dump PM values.

Dump all PM register to the console.

**Header File**

- components/perfmon/include/xtensa_perfmon_apis.h

**Functions**

**esp_err_t xtensa_perfmon_exec**(const xtensa_perfmon_config_t *config)

Execute PM.

- **config** -[in] pointer to the configuration structure

  - ESP_OK if no errors
  - ESP_ERR_INVALID_ARG if one of the required parameters not defined
  - ESP_FAIL - counter overflow

**void xtensa_perfmon_view_cb**(void *params, uint32_t select, uint32_t mask, uint32_t value)

Dump PM results.

Callback to dump perfmon result to a FILE* stream specified in perfmon_config_t::callback_params. If callback_params is set to NULL, will print to stdout.

- **params** -[in] used parameters passed from configuration (callback_params). This parameter expected as FILE* hanle, where data will be stored. If this parameter NULL, then data will be stored to the stdout.
- **select** -[in] select value for current counter
- **mask** -[in] mask value for current counter
- **value** -[in] counter value for current counter

**Structures**

**struct xtensa_perfmon_config**

Performance monitor configuration structure.

Structure to configure performance counter to measure dedicated function

**Public Members**

**int repeat_count**

how much times function will be called before the callback will be repeated
float max_deviation
    Difference between min and max counter number 0..1, 0 - no difference, 1 - not used

void *call_params
    This pointer will be passed to the call_function as a parameter

void (*call_function)(void *params)
    pointer to the function that have to be called

void (*callback)(void *params, uint32_t select, uint32_t mask, uint32_t value)
    pointer to the function that will be called with result parameters

void *callback_params
    parameter that will be passed to the callback

int tracelevel
    trace level for all counters. In case of negative value, the filter will be ignored. If it’s >=0, then the
    perfmon will count only when interrupt level > tracelevel. It’s useful to monitor interrupts.

uint32_t counters_size
    amount of counter in the list

counters_size const uint32_t *select_mask
    list of the select/mask parameters

Type Definitions

typedef struct xtensa_perfmon_config xtensa_perfmon_config_t
    Performance monitor configuration structure.
    Structure to configure performance counter to measure dedicated function

2.10.22 电源管理

概述

ESP-IDF 中集成的电源管理算法可以根据应用程序组件的需求，调整外围总线 (APB) 频率和 CPU 频率，并使芯片进入 Light-sleep 模式，尽可能减少运行应用程序的功耗。

应用程序组件可以通过创建和获取电源管理锁来控制功耗。

例如:

- 对于从 APB 获得时钟频率的外设，其驱动可以要求在使用该外设时，将 APB 频率设置为 80 MHz。
- RTOS 可以要求 CPU 在有任务准备开始运行时以最高配置频率工作。
- 一些外设可能需要中断才能启用，因此其驱动也会要求禁用 Light-sleep 模式。

请求较高的 APB 频率或 CPU 频率以及禁用 Light-sleep 模式会增加功耗，因此请将组件使用的电源管理
锁降到最少。
### 电源管理配置

编译时可使用 `CONFIG_PM_ENABLE` 选项启用电源管理功能。

启用电源管理功能将会增加中断延迟。额外延迟与多个因素有关，例如：CPU 频率、单/双核模式、是否需要进行频率切换等。CPU 频率为 240 MHz 且未启用频率调节时，最小额外延迟为 0.2 us；如果启用频率调节，且在中断人口将频率由 40 MHz 调节至 80 MHz，则最大额外延迟为 40 us。

通过调用 `esp_pm_configure()` 函数可以在应用程序中启用动态调频 (DFS) 功能和自动 Light-sleep 模式。此函数的参数 `esp_pm_config_esp32_t` 定义了频率调节的相关设置。在此参数结构中，需要初始化以下三个字段：

- `max_freq_mhz`：最大 CPU 频率 (MHz)，即获取 `ESP_PM_CPU_FREQ_MAX` 额后所使用的频率。该字段通常设置为 `CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ`。
- `min_freq_mhz`：最小 CPU 频率 (MHz)，即仅获取 `ESP_PM_APB_FREQ_MAX` 额后所使用的频率。该字段可设置为晶振 (XTAL) 频率值，或者 XTAL 频率值整数。注意，10 MHz 是生成 1 MHz 的 `REF_TICK` 默认时钟所需的最小频率。
- `light_sleep_enable`：没有获取任何管理锁时，决定系统是否需要自动进入 Light-sleep 状态 (true/false)。

如果在 `menuconfig` 中启用 `CONFIG_PM_DFS_INIT_AUTO` 选项，最大 CPU 频率将由 `CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ` 设置决定。最小 CPU 频率将锁定为 XTAL 频率。

### 备注

自动 Light-sleep 模式基于 FreeRTOSTickless Idle 方式，因此如果在 menuconfig 中没有启用 `CONFIG_FREERTOS_USE_TICKLESS_IDLE` 选项，在请求自动 Light-sleep 时，`esp_pm_configure()` 将会返回 `ESP_ERR_NOT_SUPPORTED` 错误。

例如，`EXT0` 和 `EXT1` 唤醒源可以通过 GPIO 唤醒芯片。

### 电源管理

应用程序可以通过获取或释放管理锁来控制电源管理算法。应用程序获取电源管理锁后，电源管理算法的操作将受到下面的限制，释放电源管理锁后，限制解除。

电源管理锁设有获取/释放计数器，如果已多次获取电源管理锁，则需要将电源管理锁释放相同次数以解除限制。

ESP32 支持下表中三种电源管理锁。

<table>
<thead>
<tr>
<th>电源管理锁</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ESP_PM_CPU_FREQ_MAX</code></td>
<td>请求使用 <code>esp_pm_configure()</code> 将 CPU 频率设置为最大值。ESP32 可以将该值设置为 80 MHz、160 MHz 或 240 MHz。</td>
</tr>
<tr>
<td><code>ESP_PM_APB_FREQ_MAX</code></td>
<td>请求将 APB 频率设置为最大值，ESP32 支持的最大频率为 80 MHz。</td>
</tr>
<tr>
<td><code>ESP_PM_NO_LIGHT_SLEEP</code></td>
<td>禁止自动切至 Light-sleep 模式。</td>
</tr>
</tbody>
</table>

### ESP32 电源管理算法

下表列出了启用动态调频时如何切换 CPU 频率和 APB 频率。您可以使用 `esp_pm_configure()` 或者 `CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ` 指定 CPU 最大频率。
<table>
<thead>
<tr>
<th>CPU 最高频率</th>
<th>电源管理器获取情况</th>
<th>APB 频率和 CPU 频率</th>
</tr>
</thead>
</table>
| 240      | 获取 ESP_PM_CPU_FREQ_MAX 或 ESP_PM_APB_FREQ_MAX | CPU: 240 MHz  
APB: 80 MHz |
|          | 使用 esp_pm_configure() 为二者设置最小值 |
| 160      | 获取 ESP_PM_CPU_FREQ_MAX | CPU: 160 MHz  
APB: 80 MHz |
|          | 获取 ESP_PM_APB_FREQ_MAX, 未获得 ESP_PM_CPU_FREQ_MAX | |
|          | 使用 esp_pm_configure() 为二者设置最小值 |
| 80       | 获取 ESP_PM_CPU_FREQ_MAX 或 ESP_PM_APB_FREQ_MAX | CPU: 80 MHz  
APB: 80 MHz |
|          | 使用 esp_pm_configure() 为二者设置最小值 |

如果没有获取任何管理锁，调用 esp_pm_configure() 将启动 Light-sleep 模式。Light-sleep 模式持续时间由以下因素决定：

- 处于阻塞状态的 FreeRTOS 任务数（有限超时）
- 高分辨率定时器 API 注册的计数器数量

您也可以设置 Light-sleep 模式在最近事件（任务解除阻塞，或计时器超时）之前持续多久才唤醒芯片。
为了跳过不必要的唤醒，可以将 skip_unhandled_events 选项设置为 true 来初始化 esp_timer。带有此标志的定时器不会唤醒系统，有助于减少功耗。

**动态调频和外设驱动**

启用动态调频后，APB 频率可在一个 RTOS 滴答周期内多次更改。有些外设不受 APB 频率变更的影响，但有些外设可能会出现问题。例如，Timer Group 外设定时器依赖计数器，但计数器计数的速度将随 APB 频率的变更而变更。

以下外设不受 APB 频率变更的影响：

- UART：如果 REF_TICK 或者 XTI 用作时钟源，则 UART 不受 APB 频率变更影响。请查看 uart_config_t::source_clk。
- LEDC：如果 REF_TICK 用作时钟源，则 LEDC 不受 APB 频率变更影响。请查看 ledc_timer_config() 函数。
- RMT：如果 REF_TICK 或者 XTI 被用作时钟源，则 RMT 不受 APB 频率变更影响。请查看 rmt_config_t::flags 以及 RMT_CHANNEL_FLAGS_AWARE_DFS 宏。
- GPTimer：如果 XTI 用作时钟源，则 GPTimer 不受 APB 频率变更影响。请查看 gptimer_config_t::clock_src。
- TSENS：XTAL 或 RTC_8M 用作时钟源，因此不受 APB 频率变化影响。

目前以下外设驱动程序可感知动态调频，并在调频期间使用 ESP_PM_APB_FREQ_MAX 预置：
• SPI master
• I2C
• I2S（如果APLL锁在使用中，I2S则会启用ESP_PM_NO_LIGHT_SLEEP锁）
• SDMMC

启用以下驱动程序时，将占用ESP_PM_APB_FREQ_MAX锁：

• SPI slave：从调用spi_slave_initialize()至spi_slave_free()期间。
• Ethernet：从调用esp_eth_driver_install()至esp_eth_driver_uninstall()期间。
• WiFi：从调用esp_wifi_start()至esp_wifi_stop()期间。如果启用了调制解调器睡眠模式，广播关闭时将释放此管理锁。
• TWAI：从调用twai_driver_install()至twai_driver_uninstall()期间（只有在TWAI时钟源选择为TWAI_CLK_SRC_APB的时候生效）。
• Bluetooth：从调用esp_bt_controller_enable()至esp_bt_controller_disable()期间。如果启用了蓝牙调制解调器，广播关闭时将释放此管理锁。但依然占用ESP_PM_NO_LIGHT_SLEEP锁，除非将CONFIG_BTDM_CTRL_LOW_POWER_CLOCK选项设置为“外部32 kHz晶振”。

以下外设驱动程序无法感知动态调频，应用程序需自己获取/释放管理锁：

• PCNT
• Sigma-delta
• 旧版定时器驱动（Timer Group）
• MCPWM

API 参考

Header File

• components/esp_pm/include/esp_pm.h

Functions

esp_err_t esp_pm_configure (const void *config)

Set implementation-specific power management configuration.

参数 config - pointer to implementation-specific configuration structure (e.g. esp_pm_config.esp32)

返回
• ESP_OK on success
• ESP_ERR_INVALID_ARG if the configuration values are not correct
• ESP_ERR_NOT_SUPPORTED if certain combination of values is not supported, or if CONFIG_PM_ENABLE is not enabled in sdkconfig

esp_err_t esp_pm_get_configuration (void *config)

Get implementation-specific power management configuration.

参数 config - pointer to implementation-specific configuration structure (e.g. esp_pm_config.esp32)

返回
• ESP_OK on success
• ESP_ERR_INVALID_ARG if the pointer is null

esp_err_t esp_pm_lock_create (esp_pm_lock_type_t lock_type, int arg, const char *name, esp_pm_lock_handle_t *out_handle)

Initialize a lock handle for certain power management parameter.

When lock is created, initially it is not taken. Call esp_pm_lock_acquire to take the lock.

This function must not be called from an ISR.
### API

#### esp_pm_lock_acquire

The lock is recursive, in the sense that if esp_pm_lock_acquire is called a number of times, esp_pm_lock_release has to be called the same number of times in order to release the lock.

This function may be called from an ISR.

This function is not thread-safe w.r.t. calls to other esp_pm_lock_* functions for the same handle.

**Parameters**
- `handle` - Handle obtained from esp_pm_lock_create function

**Return**
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

#### esp_pm_lock_release

The lock is recursive, in the sense that if esp_pm_lock_acquire is called a number of times, esp_pm_lock_release has to be called the same number of times in order to release the lock.

This function may be called from an ISR.

This function is not thread-safe w.r.t. calls to other esp_pm_lock_* functions for the same handle.

**Parameters**
- `handle` - Handle obtained from esp_pm_lock_create function

**Return**
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

#### esp_pm_lock_delete

The lock must be released before calling this function.

This function must not be called from an ISR.

**Parameters**
- `handle` - Handle obtained from esp_pm_lock_create function

**Return**
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle argument is NULL
- ESP_ERR_INVALID_STATE if the lock is still acquired
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig
Chapter 2. API 参考

**esp_err_t esp_pm_dump_locks** (FILE *stream)

Dump the list of all locks to stderr

This function dumps debugging information about locks created using esp_pm_lock_create to an output stream.

This function must not be called from an ISR. If esp_pm_lock_acquire/release are called while this function is running, inconsistent results may be reported.

参数 `stream`—stream to print information to; use stdout or stderr to print to the console; use fmemopen/open_memstream to print to a string buffer.

返回
- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

## Type Definitions

typedef struct esp_pm_lock *esp_pm_lock_handle_t

Opaque handle to the power management lock.

## Enumerations

enum esp_pm_lock_type_t

Power management constraints.

Values:

enumerator ESP_PM_CPU_FREQ_MAX

Require CPU frequency to be at the maximum value set via esp_pm_configure. Argument is unused and should be set to 0.

enumerator ESP_PM_APB_FREQ_MAX

Require APB frequency to be at the maximum value supported by the chip. Argument is unused and should be set to 0.

enumerator ESP_PM_NO_LIGHT_SLEEP

Prevent the system from going into light sleep. Argument is unused and should be set to 0.

## Header File

- components/esp_pm/include/esp32/pm.h

## Structures

struct esp_pm_config_esp32_t

Power management config for ESP32.

Pass a pointer to this structure as an argument to esp_pm_configure function.

## Public Members

int max_freq_mhz

Maximum CPU frequency, in MHz
int min_freq_mhz
Minimum CPU frequency to use when no locks are taken, in MHz

bool light_sleep_enable
Enter light sleep when no locks are taken

2.10.23 POSIX Threads Support

Overview

ESP-IDF is based on FreeRTOS but offers a range of POSIX-compatible APIs that allow easy porting of third party code. This includes support for common parts of the POSIX Threads “pthreads” API.

POSIX Threads are implemented in ESP-IDF as wrappers around equivalent FreeRTOS features. The runtime memory or performance overhead of using the pthreads API is quite low, but not every feature available in either pthreads or FreeRTOS is available via the ESP-IDF pthreads support.

Pthreads can be used in ESP-IDF by including standard pthread.h header, which is included in the toolchain libc. An additional ESP-IDF specific header, esp_pthread.h, provides additional non-POSIX APIs for using some ESP-IDF features with pthreads.

C++ Standard Library implementations for std::thread, std::mutex, std::condition_variable, etc. are implemented using pthreads (via GCC libstdc++). Therefore, restrictions mentioned here also apply to the equivalent C++ standard library functionality.

RTOS Integration

Unlike many operating systems using POSIX Threads, ESP-IDF is a real-time operating system with a real-time scheduler. This means that a thread will only stop running if a higher priority task is ready to run, the thread blocks on an OS synchronization structure like a mutex, or the thread calls any of the functions sleep, vTaskDelay(), or usleep.

备注：If calling a standard libc or C++ sleep function, such as usleep defined in unistd.h, then the task will only block and yield the CPU if the sleep time is longer than one FreeRTOS tick period. If the time is shorter, the thread will busy-wait instead of yielding to another RTOS task.

By default, all POSIX Threads have the same RTOS priority, but it is possible to change this by calling a custom API.

Standard features

The following standard APIs are implemented in ESP-IDF.

Refer to standard POSIX Threads documentation, or pthread.h, for details about the standard arguments and behaviour of each function. Differences or limitations compared to the standard APIs are noted below.

Thread APIs

- pthread_create() - The attr argument is supported for setting stack size and detach state only. Other attribute fields are ignored. Unlike FreeRTOS task functions, the start_routine function is allowed to return. A “detached” type thread is automatically deleted if the function returns. The default “joinable” type thread will be suspended until pthread_join() is called on it.
- pthread_join()
- pthread_detach()
- pthread_exit()
- sched_yield()
• pthread_self() - An assert will fail if this function is called from a FreeRTOS task which is not a pthread.
• pthread_equal()

Thread Attributes
• pthread_attr_init()
• pthread_attr_destroy() - This function doesn’t need to free any resources and instead resets the attr structure to defaults (implementation is same as pthread_attr_init()).
• pthread_attr_getstacksize() / pthread_attr_setstacksize()
• pthread_attr_getdetachstate() / pthread_attr_setdetachstate()

Once
• pthread_once()

Static initializer constant PTHREAD_ONCE_INIT is supported.

备注：This function can be called from tasks created using either pthread or FreeRTOS APIs

Mutexes  POSIX Mutexes are implemented as FreeRTOS Mutex Semaphores (normal type for “fast” or “error check” mutexes, and Recursive type for “recursive” mutexes). This means that they have the same priority inheritance behaviour as mutexes created with xSemaphoreCreateMutex().
• pthread_mutex_init()
• pthread_mutex_destroy()
• pthread_mutex_lock()
• pthread_mutex_timedlock()
• pthread_mutex_trylock()
• pthread_mutex_unlock()
• pthread_mutexattr_init()
• pthread_mutexattr_destroy()
• pthread_mutexattr_gettype() / pthread_mutexattr_settype()

Static initializer constant PTHREAD_MUTEX_INITIALIZER is supported, but the non-standard static initializer constants for other mutex types are not supported.

备注：These functions can be called from tasks created using either pthread or FreeRTOS APIs

Condition Variables
• pthread_cond_init() - The attr argument is not implemented and is ignored.
• pthread_cond_destroy()
• pthread_cond_signal()
• pthread_cond_broadcast()
• pthread_cond_wait()
• pthread_cond_timedwait()

Static initializer constant PTHREAD_COND_INITIALIZER is supported.

• The resolution of pthread_cond_timedwait() timeouts is the RTOS tick period (see CONFIG_FREERTOS_HZ). Timeouts may be delayed up to one tick period after the requested timeout.

备注：These functions can be called from tasks created using either pthread or FreeRTOS APIs
Chapter 2. API 参考

Read/Write Locks

- `pthread_rwlock_init()` - The `attr` argument is not implemented and is ignored.
- `pthread_rwlock_destroy()`
- `pthread_rwlock_rdlock()`
- `pthread_rwlock_wrlock()`
- `pthread_rwlock_unlock()`

Static initializer constant `PTHREAD_RWLOCK_INITIALIZER` is supported.

备注：These functions can be called from tasks created using either pthread or FreeRTOS APIs.

Thread-Specific Data

- `pthread_key_create()` - The `destr_function` argument is supported and will be called if a thread function exits normally, calls `pthread_exit()`, or if the underlying task is deleted directly using the FreeRTOS function `vTaskDelete()`.
- `pthread_key_delete()`
- `pthread_setspecific()` / `pthread_getspecific()`

备注：These functions can be called from tasks created using either pthread or FreeRTOS APIs. When calling these functions from tasks created using FreeRTOS APIs, `CONFIG_FREERTOS_TLSP_DELETION_CALLBACKS` config option must be enabled to ensure the thread-specific data is cleaned up before the task is deleted.

备注：There are other options for thread local storage in ESP-IDF, including options with higher performance. See Thread Local Storage.

Not Implemented

The `pthread.h` header is a standard header and includes additional APIs and features which are not implemented in ESP-IDF. These include:

- `pthread_cancel()` returns `ENOSYS` if called.
- `pthread_condattr_init()` returns `ENOSYS` if called.

Other POSIX Threads functions (not listed here) are not implemented and will produce either a compiler or a linker error if referenced from an ESP-IDF application. If you identify a useful API that you would like to see implemented in ESP-IDF, please open a feature request on GitHub <https://github.com/espressif/esp-idf/issues> with the details.

ESP-IDF Extensions

The API `esp_pthread_set_cfg()` defined in the `esp_pthreads.h` header offers custom extensions to control how subsequent calls to `pthread_create()` will behave. Currently, the following configuration can be set:

- Default stack size of new threads, if not specified when calling `pthread_create()` (overrides `CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT`).
- RTOS priority of new threads (overrides `CONFIG_PTHREAD_TASK_PRIO_DEFAULT`).
- Core affinity / core pinning of new threads (overrides `CONFIG_PTHREAD_TASK_CORE_DEFAULT`).
- FreeRTOS task name for new threads (overrides `CONFIG_PTHREAD_TASK_NAME_DEFAULT`).

This configuration is scoped to the calling thread (or FreeRTOS task), meaning that `esp_pthread_set_cfg()` can be called independently in different threads or tasks. If the `inherit_cfg` flag is set in the current configuration.
then any new thread created will inherit the creator’s configuration (if that thread calls `pthread_create()` recursively), otherwise the new thread will have the default configuration.

**Examples**

- `system/pthread` demonstrates using the pthreads API to create threads
- `cxx/pthread` demonstrates using C++ Standard Library functions with threads

**API Reference**

**Header File**

- `components/pthread/include/esp_pthread.h`

**Functions**

- `esp_pthread_cfg_t esp_pthread_get_default_config (void)`
  
  Creates a default pthread configuration based on the values set via menuconfig.

  返回 A default configuration structure.

- `esp_err_t esp_pthread_set_cfg (const esp_pthread_cfg_t *cfg)`

  Configure parameters for creating pthread.

  This API allows you to configure how the subsequent `pthread_create()` call will behave. This call can be used to set up configuration parameters like stack size, priority, configuration inheritance etc.

  If the ‘inherit’ flag in the configuration structure is enabled, then the same configuration is also inherited in the thread subtree.

  备注: Passing non-NULL attributes to `pthread_create()` will override the stack_size parameter set using this API

  参数 `cfg` - The pthread config parameters

  返回

  - ESP_OK if configuration was successfully set
  - ESP_ERR_NO_MEM if out of memory
  - ESP_ERR_INVALID_ARG if stack_size is less than PTHREAD_STACK_MIN

- `esp_err_t esp_pthread_get_cfg (esp_pthread_cfg_t *p)`

  Get current pthread creation configuration.

  This will retrieve the current configuration that will be used for creating threads.

  参数 `p` - Pointer to the pthread config structure that will be updated with the currently configured parameters

  返回

  - ESP_OK if the configuration was available
  - ESP_ERR_NOT_FOUND if a configuration wasn’t previously set

- `esp_err_t esp_pthread_init (void)`

  Initialize pthread library.

**Structures**

- `struct esp_pthread_cfg_t`
  
  pthread configuration structure that influences pthread creation
Public Members

`size_t stack_size`
- The stack size of the pthread.

`size_t prio`
- The thread’s priority.

`bool inherit_cfg`
- Inherit this configuration further.

`const char* thread_name`
- The thread name.

`int pin_to_core`
- The core id to pin the thread to. Has the same value range as xCoreId argument of xTaskCreatePinnedToCore.

Macros

`PTHREAD_STACK_MIN`

### 2.10.24 Random Number Generation

ESP32 contains a hardware random number generator, values from it can be obtained using the APIs `esp_random()` and `esp_fill_random()`.

The hardware RNG produces true random numbers under any of the following conditions:

- RF subsystem is enabled (i.e. Wi-Fi or Bluetooth are enabled).
- An internal entropy source has been enabled by calling `bootloader_random_enable()` and not yet disabled by calling `bootloader_random_disable()`.
- While the ESP-IDF is running. This is because the default ESP-IDF bootloader implementation calls `bootloader_random_enable()` when the bootloader starts, and `bootloader_random_disable()` before executing the app.

When any of these conditions are true, samples of physical noise are continuously mixed into the internal hardware RNG state to provide entropy. Consult the ESP32 Technical Reference Manual > Random Number Generator (RNG) [PDF] chapter for more details.

If none of the above conditions are true, the output of the RNG should be considered pseudo-random only.

Startup

During startup, ESP-IDF bootloader temporarily enables a non-RF entropy source (internal reference voltage noise) that provides entropy for any first boot key generation. However, after the app starts executing then normally only pseudo-random numbers are available until Wi-Fi or Bluetooth are initialized.

To re-enable the entropy source temporarily during app startup, or for an application that does not use Wi-Fi or Bluetooth, call the function `bootloader_random_enable()` to re-enable the internal entropy source. The function `bootloader_random_disable()` must be called to disable the entropy source again before using ADC, I2S, Wi-Fi or Bluetooth.

*备注：* The entropy source enabled during the boot process by the ESP-IDF Second Stage Bootloader will seed the internal RNG state with some entropy. However, the internal hardware RNG state is not large enough to provide a
continuous stream of true random numbers. This is why a continuous entropy source must be enabled whenever true random numbers are required.

**API Reference**

**Header File**
- components/esp_hw_support/include/esp_random.h

**Functions**

```c
uint32_t esp_random (void)
```
Get one random 32-bit word from hardware RNG.

If Wi-Fi or Bluetooth are enabled, this function returns true random numbers. In other situations, if true random numbers are required then consult the ESP-IDF Programming Guide “Random Number Generation” section for necessary prerequisites.

This function automatically busy-waits to ensure enough external entropy has been introduced into the hardware RNG state, before returning a new random number. This delay is very short (always less than 100 CPU cycles).

```c
void esp_fill_random (void *buf, size_t len)
```
Fill a buffer with random bytes from hardware RNG.

*备注:* This function is implemented via calls to esp_random(), so the same constraints apply.

**参数**
- `buf` – Pointer to buffer to fill with random numbers.
- `len` – Length of buffer in bytes

**Header File**
- components/bootloader_support/include/bootloader_random.h

**Functions**

```c
void bootloader_random_enable (void)
```
Enable an entropy source for RNG if RF subsystem is disabled.

The exact internal entropy source mechanism depends on the chip in use but all SoCs use the SAR ADC to continuously mix random bits (an internal noise reading) into the HWRNG. Consult the SoC Technical Reference Manual for more information.

Can also be called from app code, if true random numbers are required without initialized RF subsystem. This might be the case in early startup code of the application when the RF subsystem has not started yet or if the RF subsystem should not be enabled for power saving.

void bootloader_random_disable (void)
    Disable entropy source for RNG.
    Disables internal entropy source. Must be called after bootloader_random_enable() and before RF subsystem features, ADC, or I2S (ESP32 only) are initialized.
    Consult the ESP-IDF Programming Guide “Random Number Generation” section for details.

void bootloader_fill_random (void *buffer, size_t length)
    Fill buffer with ‘length’ random bytes.

备注：If this function is being called from app code only, and never from the bootloader, then it’s better to call esp_fill_random().

参数
    • buffer - Pointer to buffer
    • length - This many bytes of random data will be copied to buffer

getrandom

A compatible version of the Linux getrandom() function is also provided for ease of porting:

```c
#include <sys/random.h>

ssize_t getrandom (void *buf, size_t buflen, unsigned int flags);
```

This function is implemented by calling esp_fill_random() internally.

The flags argument is ignored, this function is always non-blocking but the strength of any random numbers is dependent on the same conditions described above.

Return value is -1 (with errno set to EFAULT) if the buf argument is NULL, and equal to buflen otherwise.

## 2.10.25 睡眠模式

概述

ESP32 具有 Light-sleep 和 Deep-sleep 两种睡眠节能模式。

在 Light-sleep 模式下，数字外设、CPU、以及大部分 RAM 都使用时钟门控，同时电源电压降低。退出该模式后，数字外设、CPU 和 RAM 恢复运行，内部状态保持不变。

在 Deep-sleep 模式下，CPU、大部分 RAM、以及所有由时钟 APB_CLK 驱动的数字外设都会被断电。芯片上继续处于供电状态的部分仅包括：

- RTC 控制器
- ULP 协处理器
- RTC 高速内存
- RTC 低速内存
Light-sleep 和 Deep-sleep 模式有多种唤醒源。这些唤醒源也可以组合在一起，此时任何一个唤醒源都可以触发唤醒。通过 API esp_sleep_enable_X_wakeup 可启用唤醒源，通过 API esp_sleep_disable_wakeup_source() 可禁用唤醒源，详见下一小节。在系统进入 Light-sleep 或 Deep-sleep 模式前，可以在任意时刻配置唤醒源。

此外，应用程序可以使用 API esp_sleep_pd_config() 强制 RTC 外设和 RTC 内存进入特定断电模式。

配置唤醒源后，应用程序就可以使用 API esp_light_sleep_start() 或 esp_deep_sleep_start() 进入睡眠模式。此时，系统将按照被请求的唤醒源配置硬件，同时 RTC 控制器会给 CPU 和数字外设断电。

如果保持 Wi-Fi 连接，请启用 Wi-Fi Modem-sleep 模式和自动 Light-sleep 模式（请参阅电源管理 API）。在这两种模式下，Wi-Fi 驱动程序发出请求时，系统将自动从睡眠中被唤醒，从而保持与 AP 的连接。

睡眠模式下的 Wi-Fi 和 Bluetooth 功能

在 Light-sleep 和 Deep-sleep 模式下，无线外设会被断电。因此，在进入这两种睡眠模式前，应用程序必须调用恰当的函数 (esp_bluedroid_disable()，esp_bt_controller_disable() 或 esp_wifi_stop()) 来禁用 Wi-Fi 和 Bluetooth。在 Light-sleep 或 Deep-sleep 模式下，即使不调用这些函数也无法连接 Wi-Fi 和 Bluetooth。

唤醒源

定时器  RTC 控制器中内嵌定时器，可用于在预定义的时间到达后唤醒芯片。时间精度为微秒，但其实时分辨率依赖于为 RTC SLOW_CLK 所选择的时钟源。

关于 RTC 时钟选项的更多细节，请参考 ESP32 技术参考手册 > ULP 协处理器 [PDF]。

在这种唤醒模式下，无需为睡眠模式中的 RTC 外设或内存供电。

调用 esp_sleep_enable_timer_wakeup() 函数可启用使用定时器唤醒睡眠模式。

触摸传感器  RTC IO 模块包含这样一个逻辑——当发生触摸传感器中断时，触发唤醒。要启用此唤醒源，用户需要在芯片进入睡眠模式前配置触摸传感器中断功能。

ESP32 修订版 0 和 1 仅在 RTC 外设没有被强制供电时支持该唤醒源（即 ESP_PD_DOMAIN_RTC_PERIPH 应被设置为 ESP_PD_OPTION_AUTO）。

可调用 esp_sleep_enable_touchpad_wakeup() 函数来启用该唤醒源。

外部唤醒 (ext0)  RTC IO 模块包含这样一个逻辑——当某个 RTC GPIO 被设置为预定义的逻辑值时，触发唤醒。RTC IO 是 RTC 外设电源域的一部分，因此如果该唤醒源被请求，RTC 外设将在 Deep-sleep 模式期间保持供电。

在此模式下，RTC IO 模块被使能，因此也可以使用内部上拉或下拉电阻。配置时，应用程序需要在调用函数 esp_deep_sleep_start() 前先调用函数 rtc_gpio_pullup_en() 和 rtc_gpio_pulldown_en()。

在 ESP32 修订版 0 和 1 中，此唤醒源与 ULP 和触摸传感器唤醒源都不兼容。

可调用 esp_sleep_enable_ext0_wakeup() 函数来启用此唤醒源。

警告：从睡眠模式中唤醒后，用于唤醒的 IO pad 将被配置为 RTC IO。因此，在将该 pad 用作数字 GPIO 之前，请调用 rtc_gpio_deinit() 函数对其进行重新配置。
外部唤醒 (ext)  
RTC 控制器中包含使用多个 RTC GPIO 触发唤醒的逻辑。您可以从以下两个逻辑函数中选择其一，用于触发唤醒：
- 当任意一个所选管脚为高电平时唤醒 (ESP_EXT1_WAKEUP_ANY_HIGH)
- 当所有所选管脚为低电平时唤醒 (ESP_EXT1_WAKEUP_ALL_LOW)

此唤醒源由 RTC 控制器实现。这种模式下的 RTC 外设和 RTC 内存可以被断电，但如果 RTC 外设被断电，则需要在 RTC 外设电源域在睡眠期间保持开启，并在进入睡眠前使用函数 `esp_sleep_pd_config()` 配置上拉或下拉电阻。

```c
esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_ON);
gpio_pullup_dis(gpio_num);gpio_pulldown_en(gpio_num);
```

警告：从睡眠模式中唤醒后，用于唤醒的 IO pad 将被配置为 RTC IO。因此在将该 pad 用作数字 GPIO 前，请调用 `rtc_gpio_deinit()` 函数对其进行重新配置。

可调用 `esp_sleep_enable_ext1_wakeup()` 函数来启用此唤醒源。

ULP 协处理器唤醒 当芯片处于睡眠模式时，ULP 协处理器仍然运行，可用于唤醒传感器、监视 ADC 或触碰传感器的值，并在检测到特殊事件时唤醒芯片。ULP 协处理器是 RTC 外设电源域的一部分，运行在 RTC 低速内存中的程序。如果这一唤醒源被请求，RTC 低速内存将会在睡眠期间保持供电状态。

RTC 外设会在 ULP 协处理器开始运行前自动上电；一旦程序停止运行，RTC 外设会再次自动断电。

ESP32 修订版 0 和 1 仅在 RTC 外设没有被强制供电时支持该唤醒（即 ESP_PD_DOMAIN_RTC_PERIPH 应被设置为 ESP_PD_OPTION_AUTO）。

可调用 `esp_sleep_enable_ulp_wakeup()` 函数来启用此唤醒源。

GPIO 唤醒（仅适用于 Light-sleep 模式） 除了上述 EX0 和 EX1 唤醒源之外，还有一种从外部唤醒 Light-sleep 模式的方法——使用函数 `gpio_pullup_dis()`。启用该唤醒源后，可将每个管脚单独配置为高电平或低电平时唤醒。EX0 和 EX1 唤醒源只能用于 RTC IO，但此唤醒源既可以用于 RTC IO，也可以用于数字 IO。

可调用 `esp_sleep_enable_gpio_wakeup()` 函数来启用此唤醒源。

警告：在进入 Light-sleep 模式前，请查看您将要驱动的 GPIO 管脚的电源域。如果有管脚属于 VDD_SDIO 电源域，必须将此电源域配置为在睡眠期间保持供电。

例如，在 ESP32-WROOM-32 开发板上，GPIO16 和 GPIO17 连接到 VDD_SDIO 电源域。如果这两个管脚被配置为在睡眠期间保持高电平，按需需将对应电源域配置为保持供电。您可以使用函数 `esp_sleep_pd_config()`：

```c
esp_sleep_pd_config(ESP_PD_DOMAIN_VDDSDIO, ESP_PD_OPTION_ON);
```

UART 唤醒（仅适用于 Light-sleep 模式） 当 ESP32 从外部设备接收 UART 输入时，通常需要在输入数据可用时唤醒芯片。UART 外设支持在 RX 管脚上检测到一定数量的上升沿时，将芯片从 Light-sleep 模式中唤醒。调用 `uart_set_wakeup_threshold()` 函数可设置检测上升沿的数量。请注意，触发唤醒的字符（及该字符前的任何字符）在唤醒后不会被 UART 接收，因此在发送数据之前，外部设备通常需要首先向 ESP32 额外发送一个字符以触发唤醒。

可调用 `esp_sleep_enable_uart_wakeup()` 函数来启用此唤醒源。

RTC 外设和内存断电

默认情况下，调用函数 `esp_deep_sleep_start()` 和 `esp_light_sleep_start()` 后，所有唤醒源不需要的 RTC 电源域都会被断电。可调用函数 `esp_sleep_pd_config()` 来修改这一设置。

Espressif Systems 2040 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
注意：在 ESP32 修订版 1 中，RTC 高速内存存在 Deep-sleep 期间将总是保持使能，以保证复位后可运行 Deep-sleep stub。如果应用程序在 Deep-sleep 模式后无需复位，您也可以对其进行修改。

如果程序中的某些值被放入 RTC 低速内存中（例如使用 RTC_DATA_ATTR 属性），RTC 低速内存将默认保持供电。如果有需要，也可以使用函数 esp_sleep_pd_config() 对其进行修改。

**Flash 断电**

默认情况下，调用函数 esp_light_sleep_start() 后，flash 不会断电，因为在 sleep 过程中断电 flash 存在风险。具体而言，flash 断电需要时间，但是在此期间，系统有可能被唤醒，导致 flash 重新被上电。此时，断电尚未完成又重新上电的硬件行为有可能导致 flash 无法正常工作。

理论上讲，在 flash 完全断电后可以仅唤醒系统。然而，实际情况是 flash 断电所需的时间很难预测。如果用户为 flash 供电电路添加了滤波电容，断电所需时间可能会更长。此外，即使可以预知 flash 彻底断电所需的时间，有时也不能通过设置足够长的睡眠时间来确保 flash 断电的安全（比如，突发的异步唤醒源会使得实际的睡眠时间不可控）。

**警告：**如果在 flash 的供电电路中添加了滤波电容，那么应该尽一切可能避免 flash 断电。

因为这些不可控的因素，ESP-IDF 很难保证 flash 断电的绝对安全。因此 ESP-IDF 不推荐用户断电 flash。对于一些功耗敏感型应用，可以通过设置 Kconfig 配置项 CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND 来减少 light sleep 期间 flash 的功耗。这种模式在几乎所有场合下都要比断电 flash 更好，兼顾了安全性和功耗。

值得一提的是，PSRAM 也有一个类似的 Kconfig 配置项 CONFIG_ESP_SLEEP_PSRAM_LEAKAGE_WORKAROUND。

考虑到有些用户能够充分评估断电 flash 的风险，并希望通过断电 flash 来获得更低的功耗，因此 ESP-IDF 提供了两种断电 flash 的机制：

- 设置 Kconfig 配置项 CONFIG_ESP_SLEEP_POWER_DOWN_FLASH 将使 ESP-IDF 以一个严格的条件来断电 flash。严格的条件具体指的是，RTC timer 是唯一的唤醒源。睡眠时间比 flash 彻底断电所需时间更长。
- 调用函数 esp_sleep_pd_config(ESP_PD_DOMAIN_VDDSDIO, ESP_PD_OPTION_OFF) 将使 ESP-IDF 以一个宽松的条件来断电 flash。宽松的条件具体指的是 RTC timer 唤醒源未被使能或睡眠时间比 flash 彻底断电所需时间更长。

**备注：**

- Light sleep 时，ESP-IDF 没有提供保证 flash 一定会断电的机制。
- 不管用户的配置如何，函数 esp_deep_sleep_start() 都会强制断电 flash。

**进入 Light-sleep 模式**

函数 esp_light_sleep_start() 可用于在配置唤醒源后进入 Light-sleep 模式，也可用于在未配置唤醒源的情况下进入 Light-sleep 模式。在后一种情况下，芯片将一直处于睡眠模式，直到从外部被复位。

**进入 Deep-sleep 模式**

函数 esp_deep_sleep_start() 可用于在配置唤醒源后进入 Deep-sleep 模式，也可用于在未配置唤醒源的情况下进入 Deep-sleep 模式。在后一种情况下，芯片将一直处于睡眠模式，直到从外部被复位。
配置 IO

一些 ESP32 IO 在默认情况下启用内部上拉或下拉电阻。如果这些管脚在 Deep-sleep 模式下中受外部电路驱动，电流会经过这些上下拉电阻时，可能会增加电流消耗。

想要隔离这些管脚以避免额外的电流消耗，请调用 `rtc_gpio_isolate()` 函数。

例如，在 ESP32-WROVER 模组上，GPIO12 在外部上拉，但其在 ESP32 芯片中也有内部下拉。这意味着在 Deep-sleep 模式中，电流会由这些外部和内部电阻，使电流消耗超出可能的最小值。

在函数 `esp_deep_sleep_start()` 前增加以下代码即可避免额外电流消耗：

```c
rtc_gpio_isolate(GPIO_NUM_12);
```

UART 输出处理

在进入睡眠模式之前，调用函数 `esp_deep_sleep_start()` 会冲刷掉 UART FIFO 缓存。

当使用函数 `esp_light_sleep_start()` 进入 Light-sleep 模式时，UART FIFO 将不会被冲刷。与之相反，UART 输出将被暂停。FIFO 中的剩余字符将在 Light-sleep 唤醒后被发送。

检查睡眠唤醒原因

`esp_sleep_get_wakeup_cause()` 函数可用于检测是何种唤醒源在睡眠期间被触发。

对于触摸传感器唤醒源，可以调用函数 `esp_sleep_get_touchpad_wakeup_status()` 来确认触发唤醒的触摸管脚。

对于 `ext1` 唤醒源，可以调用函数 `esp_sleep_get_ext1_wakeup_status()` 来确认触发唤醒的触摸管脚。

禁用睡眠模式唤醒源

调用 API `esp_sleep_disable_wakeup_source()` 可以禁用给定唤醒源的触发器，从而禁用该唤醒源。此外，如果将参数设置为 `ESP_SLEEP_WAKEUP_ALL`，该函数可用于禁用所有触发器。

应用程序示例

- `protocols/sntp`：如何实现 Deep-sleep 模式的基本功能，周期性唤醒 ESP 模块，以从 NTP 服务器获取时间。
- `wifi/power_save`：如何实现 Modem-sleep 模式。
- `system/deep_sleep`：如何使用 Deep-sleep 唤醒触发器和 ULP 协处理器编程。

API 参考

Header File

- `components/esp_hw_support/include/esp_sleep.h`

Functions

`esp_err_t esp_sleep_disable_wakeup_source(esp_sleep_source_t source)`

Disable wakeup source.

This function is used to deactivate wake up trigger for source defined as parameter of the function.

See docs/sleep-modes.rst for details.
**API**

**esp_sleep_enable_ulp_wakeup**(void)

Enable wakeup by ULP coprocessor.

**esp_sleep_enable_timer_wakeup**(uint64_t time_in_us)

Enable wakeup by timer.

**esp_sleep_enable_touchpad_wakeup**(void)

Enable wakeup by touch sensor.

**esp_sleep_get_touchpad_wakeup_status**(void)

Get the touch pad which caused wakeup.

If wakeup was caused by another source, this function will return TOUCH_PAD_MAX;

**esp_sleep_is_valid_wakeup_gpio**(gpio_num_t gpio_num)

Returns true if a GPIO number is valid for use as wakeup source.
For SoCs with RTC IO capability, this can be any valid RTC IO input pin.

**Parameters**
- `gpio_num` - Number of the GPIO to test for wakeup source capability
- `level` - Input level which will trigger wakeup (0=low, 1=high)

**Return**
- ESP_OK on success
- ESP_ERR_INVALID_ARG if the selected GPIO is not an RTC GPIO, or the mode is invalid
- ESP_ERR_INVALID_STATE if wakeup triggers conflict

**esp_err_t esp_sleep_enable_ext0_wakeup (gpio_num_t gpio_num, int level)**

Enable wakeup using a pin.

This function uses external wakeup feature of RTC IO peripheral. It will work only if RTC peripherals are kept on during sleep.

This feature can monitor any pin which is an RTC IO. Once the pin transitions into the state given by level argument, the chip will be woken up.

**Note:** This function does not modify pin configuration. The pin is configured in `esp_deep_sleep_start/esp_light_sleep_start`, immediately before entering sleep mode.

**Note:** ESP32: ext0 wakeup source cannot be used together with touch or ULP wakeup sources.

**Parameters**
- `gpio_num` - GPIO number used as wakeup source. Only GPIOs with the RTC functionality can be used. For different SoCs, the related GPIOs are:
  - ESP32: 0, 2, 4, 12-15, 25-27, 32-39;
  - ESP32-S2: 0-21;
  - ESP32-S3: 0-21.
- `level` - Input level which will trigger wakeup (0=low, 1=high)

**Return**
- ESP_OK on success

**esp_err_t esp_sleep_enable_ext1_wakeup (uint64_t mask, esp_sleep_ext1_wakeup_mode_t mode)**

Enable wakeup using multiple pins.

This function uses external wakeup feature of RTC controller. It will work even if RTC peripherals are shut down during sleep.

This feature can monitor any number of pins which are in RTC IOs. Once any of the selected pins goes into the state given by mode argument, the chip will be woken up.

**Note:** This function does not modify pin configuration. The pins are configured in `esp_deep_sleep_start/esp_light_sleep_start`, immediately before entering sleep mode.

**Note:** Internal pullups and pulldowns don’t work when RTC peripherals are shut down. In this case, external resistors need to be added. Alternatively, RTC peripherals (and pullups/pulldowns) may be kept enabled using `esp_sleep_pd_config` function.

**Parameters**
- `mask` - Bit mask of GPIO numbers which will cause wakeup. Only GPIOs which have RTC functionality can be used in this bit map. For different SoCs, the related GPIOs are:
  - ESP32: 0, 2, 4, 12-15, 25-27, 32-39;
Chapter 2. API

– ESP32-S2: 0-21;

**mode** select logic function used to determine wakeup condition:
– ESP_EXT1_WAKEUP_ALL_LOW: wake up when all selected GPIOs are low
– ESP_EXT1_WAKEUP_ANY_HIGH: wake up when any of the selected GPIOs is high

**esp_err_t esp_sleep_enable_gpio_wakeup (void)**
Enable wakeup from light sleep using GPIOs.

Each GPIO supports wakeup function, which can be triggered on either low level or high level. Unlike EXT0
and EXT1 wakeup sources, this method can be used both for all IOs: RTC IOs and digital IOs. It can only be
used to wakeup from light sleep though.

To enable wakeup, first call gpio_wakeup_enable, specifying gpio number and wakeup level, for each GPIO
which is used for wakeup. Then call this function to enable wakeup feature.

**备注:** On ESP32, GPIO wakeup source can not be used together with touch or ULP wakeup sources.

**esp_err_t esp_sleep_enable_uart_wakeup (int uart_num)**
Enable wakeup from light sleep using UART.

Use uart_set_wakeup_threshold function to configure UART wakeup threshold.

Wakeup from light sleep takes some time, so not every character sent to the UART can be received by the
application.

**备注:** ESP32 does not support wakeup from UART2.

**参数** uart_num – UART port to wake up from

**返回**
– ESP_OK on success
– ESP_ERR_INVALID_ARG if wakeup from given UART is not supported

**esp_err_t esp_sleep_enable_bt_wakeup (void)**
Enable wakeup by bluetooth.

**返回**
– ESP_OK on success
– ESP_ERR_NOT_SUPPORTED if wakeup from bluetooth is not supported

**esp_err_t esp_sleep_disable_bt_wakeup (void)**
Disable wakeup by bluetooth.

**返回**
– ESP_OK on success
– ESP_ERR_NOT_SUPPORTED if wakeup from bluetooth is not supported

**esp_err_t esp_sleep_enable_wifi_wakeup (void)**
Enable wakeup by WiFi MAC.
Chapter 2. API

Return
- ESP_OK on success

`esp_err_t esp_sleep_disable_wifi_wakeup(void)`
Disable wakeup by WiFi MAC.

Return
- ESP_OK on success

`uint64_t esp_sleep_get_ext1_wakeup_status(void)`
Get the bit mask of GPIOs which caused wakeup (ext1)
If wakeup was caused by another source, this function will return 0.

Return bit mask, if GPIO caused wakeup, BIT(n) will be set

`esp_err_t esp_sleep_pd_config(esp_sleep_pd_domain_t domain, esp_sleep_pd_option_t option)`
Set power down mode for an RTC power domain in sleep mode.
If not set set using this API, all power domains default to ESP_PD_OPTION_AUTO.

**Parameter**
- domain - power domain to configure
- option - power down option (ESP_PD_OPTION_OFF, ESP_PD_OPTION_ON, or ESP_PD_OPTION_AUTO)

Return
- ESP_OK on success
- ESP_ERR_INVALID_ARG if either of the arguments is out of range

`void esp_deep_sleep_start(void)`
Enter deep sleep with the configured wakeup options.
This function does not return.

`esp_err_t esp_light_sleep_start(void)`
Enter light sleep with the configured wakeup options.

Return
- ESP_OK on success (returned after wakeup)
- ESP_ERR_SLEEP_REJECT sleep request is rejected (wakeup source set before the sleep request)
- ESP_ERR_SLEEP_TOO_SHORT_SLEEP_DURATATION after deducting the sleep flow overhead, the final sleep duration is too short to cover the minimum sleep duration of the chip, when rtc timer wakeup source enabled

`void esp_deep_sleep(uint64_t time_in_us)`
Enter deep-sleep mode.
The device will automatically wake up after the deep-sleep time. Upon waking up, the device calls deep sleep wake stub, and then proceeds to load application.

Call to this function is equivalent to a call to esp_deep_sleep_enable_timer_wakeup followed by a call to esp_deep_sleep_start.

esp_deep_sleep does not shut down WiFi, BT, and higher level protocol connections gracefully. Make sure relevant WiFi and BT stack functions are called to close any connections and deinitialize the peripherals. These include:
- esp_bluedroid_disable
- esp_bt_controller_disable
- esp_wifi_stop

This function does not return.

**Note:** The device will wake up immediately if the deep-sleep time is set to 0
参数 `time_in_us`—deep-sleep time, unit: microsecond

```c
void esp_sleep_wakeup_cause_t esp_sleep_get_wakeup_cause()
```

Get the wakeup source which caused wakeup from sleep.

返回 cause of wake up from last sleep (deep sleep or light sleep)

```c
void esp_wake_deep_sleep()
```

Default stub to run on wake from deep sleep.

- Allows for executing code immediately on wake from sleep, before the software bootloader or ESP-IDF app has started up.
- This function is weak-linked, so you can implement your own version to run code immediately when the chip wakes from sleep.
- See docs/deep-sleep-stub.rst for details.

```c
void esp_set_deep_sleep_wake_stub(esp_deep_sleep_wake_stub_fn_t new_stub)
```

Install a new stub at runtime to run on wake from deep sleep.

- If implementing esp_wake_deep_sleep() then it is not necessary to call this function.
- However, it is possible to call this function to substitute a different deep sleep stub. Any function used as a deep sleep stub must be marked RTC_IRAM_ATTR, and must obey the same rules given for esp_wake_deep_sleep().

```c
esp_deep_sleep_wake_stub_fn_t esp_get_deep_sleep_wake_stub()
```

Get current wake from deep sleep stub.

返回 Return current wake from deep sleep stub, or NULL if no stub is installed.

```c
void esp_default_wake_deep_sleep()
```

The default esp-idf-provided esp_wake_deep_sleep() stub.

- See docs/deep-sleep-stub.rst for details.

```c
void esp_deep_sleep_disable_rom_logging()
```

Disable logging from the ROM code after deep sleep.

- Using LSB of RTCSTORE4.

```c
void esp_sleep_config_gpio_isolate()
```

Configure to isolate all GPIO pins in sleep state.

```c
void esp_sleep_enable_gpio_switch(bool enable)
```

Enable or disable GPIO pins status switching between slept status and waked status.

参数 `enable`—decide whether to switch status or not

Type Definitions

typedef `esp_sleep_source_t` `esp_sleep_wakeup_cause_t`

typedef void (*`esp_deep_sleep_wake_stub_fn_t`)(void)

Function type for stub to run on wake from sleep.

Enumerations

enum `esp_sleep_ext1_wakeup_mode_t`

Logic function used for EXT1 wakeup mode.

Values:
enumerator **ESP_EXT1_WAKEUP_ALL_LOW**
  Wake the chip when all selected GPIOs go low.

enumerator **ESP_EXT1_WAKEUP_ANY_HIGH**
  Wake the chip when any of the selected GPIOs go high.

enum **esp_sleep_pd_domain_t**
  Power domains which can be powered down in sleep mode.
  
  *Values:*

  enumerator **ESP_PD_DOMAIN_RTC_PERIPH**
  RTC IO, sensors and ULP co-processor.

  enumerator **ESP_PD_DOMAIN_RTC_SLOW_MEM**
  RTC slow memory.

  enumerator **ESP_PD_DOMAIN_RTC_FAST_MEM**
  RTC fast memory.

  enumerator **ESP_PD_DOMAIN_XTAL**
  XTAL oscillator.

  enumerator **ESP_PD_DOMAIN_RTC8M**
  Internal 8M oscillator.

  enumerator **ESP_PD_DOMAIN_VDDSDIO**
  VDD_SDIO.

  enumerator **ESP_PD_DOMAIN_MAX**
  Number of domains.

enum **esp_sleep_pd_option_t**
  Power down options.
  
  *Values:*

  enumerator **ESP_PD_OPTION_OFF**
  Power down the power domain in sleep mode.

  enumerator **ESP_PD_OPTION_ON**
  Keep power domain enabled during sleep mode.

  enumerator **ESP_PD_OPTION_AUTO**
  Keep power domain enabled in sleep mode, if it is needed by one of the wakeup options. Otherwise power it down.

enum **esp_sleep_source_t**
  Sleep wakeup cause.
  
  *Values:*

  enumerator **ESP_SLEEP_SRC_PBTN**
  Button press.

  enumerator **ESP_SLEEP_SRC_WAKEUP**
  Alternate wakeup.

  enumerator **ESP_SLEEP_SRC_RTC**
  RTC wakeup.

  enumerator **ESP_SLEEP_SRC_MAX**
  Number of sources.
enumerator `ESP_SLEEP_WAKEUP_UNDEFINED`
In case of deep sleep, reset was not caused by exit from deep sleep.

enumerator `ESP_SLEEP_WAKEUP_ALL`
Not a wakeup cause, used to disable all wakeup sources with esp_sleep_disable_wakeup_source.

enumerator `ESP_SLEEP_WAKEUP_EXT0`
Wakeup caused by external signal using RTC_IO.

enumerator `ESP_SLEEP_WAKEUP_EXT1`
Wakeup caused by external signal using RTC_CNTL.

enumerator `ESP_SLEEP_WAKEUP_TIMER`
Wakeup caused by timer.

enumerator `ESP_SLEEP_WAKEUP_TOUCHPAD`
Wakeup caused by touchpad.

enumerator `ESP_SLEEP_WAKEUP_ULP`
Wakeup caused by ULP program.

enumerator `ESP_SLEEP_WAKEUP_GPIO`
Wakeup caused by GPIO (light sleep only on ESP32, S2 and S3)

enumerator `ESP_SLEEP_WAKEUP_UART`
Wakeup caused by UART (light sleep only)

enumerator `ESP_SLEEP_WAKEUP_WIFI`
Wakeup caused by WIFI (light sleep only)

enumerator `ESP_SLEEP_WAKEUP_COCPU`
Wakeup caused by COCPU int.

enumerator `ESP_SLEEP_WAKEUP_COCPU_TRAP_TRIG`
Wakeup caused by COCPU crash.

enumerator `ESP_SLEEP_WAKEUP_BT`
Wakeup caused by BT (light sleep only)

enum [anonymous]
Values:

enumerator `ESP_ERR_SLEEP_REJECT`

enumerator `ESP_ERR_SLEEP_TOO_SHORT_SLEEP_DURATION`
2.10.26  SoC Capabilities

This section lists definitions of the ESP32’s SoC hardware capabilities. These definitions are commonly used in IDF to control which hardware dependent features are supported and thus compiled into the binary.

备注：These defines are currently not considered to be part of the public API, and may be changed at any time.

API Reference

Header File

- components/soc/esp32/include/soc/soc_caps.h

Macros

SOC_CAPS_ECO_VER_MAX
SOC_ADC_SUPPORTED
SOC_DAC_SUPPORTED
SOC_MCPWM_SUPPORTED
SOC_SDMMC_HOST_SUPPORTED
SOC_BT_SUPPORTED
SOC_PCNT_SUPPORTED
SOC_WIFI_SUPPORTED
SOC_SDIO_SLAVE_SUPPORTED
SOC_TWAI_SUPPORTED
SOC_EMAC_SUPPORTED
SOC_ULP_SUPPORTED
SOC_CCOMP_TIMER_SUPPORTED
SOC_RTC_FAST_MEM_SUPPORTED
SOC_RTC_SLOW_MEM_SUPPORTED
SOC_RTC_MEM_SUPPORTED
SOC_I2S_SUPPORTED
SOC_RMT_SUPPORTED
SOC_SDM_SUPPORTED
SOC_LEDC_SUPPORTED
SOC_I2C_SUPPORTED
SOC_SUPPORT_COEXISTENCE
SOC_AES_SUPPORTED
SOC_MPI_SUPPORTED
SOC_SHA_SUPPORTED
SOC_FLASH_ENC_SUPPORTED
SOC_SECURE_BOOT_SUPPORTED
SOC_TOUCH_SENSOR_SUPPORTED
SOC_DPORT_WORKAROUND
SOC_DPORT_WORKAROUND_DIS_INTERRUPT_LVL
SOC_XTAL_SUPPORT_26M
SOC_XTAL_SUPPORT_40M
SOC_XTAL_SUPPORT_AUTO_DETECT
SOC_ADC_RTC_CTRL_SUPPORTED
< SAR ADC Module
SOC_ADC_DIG_CTRL_SUPPORTED
SOC_ADC_DMA_SUPPORTED
SOC_ADC_PERIPH_NUM
SOC_ADC_CHANNEL_NUM (PERIPH_NUM)
SOC_ADC_MAX_CHANNEL_NUM
Chapter 2. API

SOC_ADC_ATTEN_NUM
Digital

SOC_ADC_DIGI_CONTROLLER_NUM

SOC_ADC_PATT_LEN_MAX

SOC_ADC_DIGI_MIN_BITWIDTH

SOC_ADC_DIGI_MAX_BITWIDTH

SOC_ADC_DIGI_RESULT_BYTES

SOC_ADC_DIGI_DATA_BYTES_PER_CONV

SOC_ADC_SAMPLE_FREQ_THRES_HIGH

RTC
SOC_ADC_SAMPLE_FREQ_THRES_LOW

SOC_ADC_RTC_MIN_BITWIDTH

SOC_ADC_RTC_MAX_BITWIDTH

SOC_SHARED_IDCACHE_SUPPORTED

SOC_MMU_LINEAR_ADDRESS_REGION_NUM

SOC_CPU_CORES_NUM

SOC_CPU_INTR_NUM

SOC_CPU_HAS_FPU

SOC_CPU_BREAKPOINTS_NUM

SOC_CPU_WATCHPOINTS_NUM

SOC_CPU_WATCHPOINT_SIZE

SOC_DAC_CHAN_NUM

SOC_DAC_RESOLUTION

SOC_DAC_DMA_16BIT_ALIGN
SOC_GPIO_PORT
SOC_GPIO_PIN_COUNT
SOC_GPIO_VALID_GPIO_MASK
SOC_GPIO_VALID_OUTPUT_GPIO_MASK
SOC_GPIO_VALID_DIGITAL_IO_PAD_MASK
SOC_GPIO_SUPPORT_SLP_SWITCH
SOC_I2C_NUM
SOC_I2C_FIFO_LEN
   I2C hardware FIFO depth
SOC_I2C_SUPPORT_SLAVE
SOC_I2C_SUPPORT_APB
SOC_I2S_NUM
SOC_I2S_HW_VERSION_1
SOC_I2S_SUPPORTS_APLL
SOC_I2S_SUPPORTS_PDM
SOC_I2S_SUPPORTS_PDM_TX
SOC_I2S_PDM_MAX_TX_LINES
SOC_I2S_SUPPORTS_PDM_RX
SOC_I2S_PDM_MAX_RX_LINES
SOC_I2S_SUPPORTS_ADC_DAC
SOC_I2S_SUPPORTS_ADC
SOC_I2S_SUPPORTS_DAC
SOC_I2S_SUPPORTS_LCD_CAMERA
Chapter 2. API 参考

SOC_I2S_TRANS_SIZE_ALIGN_WORD

SOC_I2S_LCD_I80_VARIANT

SOC_LCD_I80_SUPPORTED
   Intel 8080 LCD is supported

SOC_LCD_I80_BUSES
   Both I2S0/1 have LCD mode

SOC_LCD_I80_BUS_WIDTH
   Intel 8080 bus width

SOC_LEDC_HAS_TIMER_SPECIFIC_MUX

SOC_LEDC_SUPPORT_APB_CLOCK

SOC_LEDC_SUPPORT_REF_TICK

SOC_LEDC_SUPPORT_HS_MODE

SOC_LEDC_CHANNEL_NUM

SOC_LEDC_TIMER_BIT_WIDE_NUM

SOC_MCPWM_GROUPS
   2 MCPWM groups on the chip (i.e., the number of independent MCPWM peripherals)

SOC_MCPWM_TIMERS_PER_GROUP
   The number of timers that each group has.

SOC_MCPWM_OPERATORS_PER_GROUP
   The number of operators that each group has.

SOC_MCPWM_COMPARATORS_PER_OPERATOR
   The number of comparators that each operator has.

SOC_MCPWM_GENERATORS_PER_OPERATOR
   The number of generators that each operator has.

SOC_MCPWM_TRIGGERS_PER_OPERATOR
   The number of triggers that each operator has.

SOC_MCPWM_GPIO_FAULTS_PER_GROUP
   The number of GPIO fault signals that each group has.

SOC_MCPWM_CAPTURE_TIMERS_PER_GROUP
   The number of capture timers that each group has.
Chapter 2. API 参考

SOC_MCPWM_CAPTURE_CHANNELS_PER_TIMER
The number of capture channels that each capture timer has.

SOC_MCPWM_GPIO_SYNCHROS_PER_GROUP
The number of GPIO synchros that each group has.

SOC_MCPWM_CLK_SUPPORT_PLL160M
Support PLL160M as clock source.

SOC_MPU_CONFIGURABLE_REGIONS_SUPPORTED

SOC_MPU_MIN_REGION_SIZE

SOC_MPU_REGIONS_MAX_NUM

SOC_MPU_REGION_RO_SUPPORTED

SOC_MPU_REGION_WO_SUPPORTED

SOC_PCNT_GROUPS

SOC_PCNT_UNITS_PER_GROUP

SOC_PCNT_CHANNELS_PER_UNIT

SOC_PCNT_THRESH_POINTS_PER_UNIT

SOC_RMT_GROUPS
One RMT group

SOC_RMT_TX_CANDIDATES_PER_GROUP
Number of channels that capable of Transmit in each group

SOC_RMT_RX_CANDIDATES_PER_GROUP
Number of channels that capable of Receive in each group

SOC_RMT_CHANNELS_PER_GROUP
Total 8 channels

SOC_RMT_MEM_WORDS_PER_CHANNEL
Each channel owns 64 words memory

SOC_RMT_SUPPORT_REF_TICK
Support set REF_TICK as the RMT clock source

SOC_RMT_SUPPORT_APB
Support set APB as the RMT clock source
SOC_RMT_CHANNEL_CLK_INDEPENDENT
    Can select different source clock for each channel

SOC_RTCIO_PIN_COUNT

SOC_RTCIO_INPUT_OUTPUT_SUPPORTED

SOC_RTCIO_HOLD_SUPPORTED

SOC_RTCIO_WAKE_SUPPORTED

SOC_SDM_GROUPS

SOC_SDM_CHANNELS_PER_GROUP

SOC_SPI HD_BOTH_INOUT_SUPPORTED

SOC_SPI AS_CS_SUPPORTED

SOC_SPI_PERIPH_NUM

SOC_SPI_DMA_CHAN_NUM

SOC_SPI_PERIPH_CS_NUM

SOC_SPI_MAX_CS_NUM

SOC_SPI_MAXIMUM BUFFER_SIZE

SOC_SPI_MAX_PRE_DIVIDER

SOC_MEMSPI_SRC_FREQ_80M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_40M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_26M_SUPPORTED

SOC_MEMSPI_SRC_FREQ_20M_SUPPORTED

SOC_SPI_PERIPH_SUPPORT_MULTILINE_MODE (spi_host)

SOC_TIMER_GROUPS

SOC_TIMER_GROUP_TIMERS_PER_GROUP

SOC_TIMER_GROUP_COUNTER_BIT_WIDTH
Chapter 2. API 参考

SOC_TIMER_GROUP_TOTAL_TIMERS

SOC_TIMER_GROUP_SUPPORT_APB

SOC_TOUCH_VERSION_1
  Hardware version of touch sensor

SOC_TOUCH_SENSOR_NUM

SOC_TOUCH_PAD_MEASURE_WAIT_MAX
  The timer frequency is 8Mhz, the max value is 0xff

SOC_TOUCH_PAD_THRESHOLD_MAX
  If set touch threshold max value, the touch sensor can’t be in touched status

SOC_TWAI_CONTROLLER_NUM

SOC_TWAI_BRP_MIN

SOC_TWAI_BRP_MAX

SOC_TWAI_CLK_SUPPORT_APB

SOC_TWAI_SUPPORT_MULTI_ADDRESS_LAYOUT

SOC_UART_NUM

SOC_UART_SUPPORT_APB_CLK
  Support APB as the clock source

SOC_UART_SUPPORT_REF_TICK
  Support REF_TICK as the clock source

SOC_UART_FIFO_LEN
  The UART hardware FIFO length

SOC_UART_BITRATE_MAX
  Max bitrate supported by UART

SOC_SPIRAM_SUPPORTED

SOC_SPI_MEM_SUPPORT_CONFIG_GPIO_BY_EFUSE

SOC_SHA_SUPPORT_PARALLEL_ENG

SOC_SHA_SUPPORT_SHA1
Chapter 2. API Reference

- SOC_SHA_SUPPORT_SHA256
- SOC_SHA_SUPPORT_SHA384
- SOC_SHA_SUPPORT_SHA512
- SOC_RSA_MAX_BIT_LEN
- SOC_AES_SUPPORT_AES_128
- SOC_AES_SUPPORT_AES_192
- SOC_AES_SUPPORT_AES_256
- SOC_SECURE_BOOT_V1
- SOC_EFUSE_SECURE_BOOT_KEY_DIGESTS
- SOC_FLASH_ENCRYPTED_XTS_AES_BLOCK_MAX
- SOC_PHY_DIG_REGS_MEM_SIZE
- SOC_PM_SUPPORT_EXT_WAKEUP
- SOC_PM_SUPPORT_RTC_PERIPH_PD
- SOC_PM_SUPPORT_RTC_FAST_MEM_PD
- SOC_PM_SUPPORT_RTC_SLOW_MEM_PD
- SOC_CLK_APLL_SUPPORTED
- SOC_APLL_MULTIPLIER_OUT_MIN_HZ
- SOC_APLL_MULTIPLIER_OUT_MAX_HZ
- SOC_APLL_MIN_HZ
- SOC_APLL_MAX_HZ
- SOC_CLK_RC_FAST_D256_SUPPORTED

Supports waking up from touch pad trigger
Chapter 2. API 参考

SOC_RTC_SLOW_CLK_SUPPORT_RC_FAST_D256
SOC_SDMMC_USE_IOMUX
SOC_SDMMC_NUM_SLOTS
SOC_WIFI_FTM_SUPPORT
    FTM is not supported
SOC_WIFI_GCMP_SUPPORT
    GCMP is not supported (GCMP128 and GCMP256)
SOC_WIFI_WAPI_SUPPORT
    Support WAPI
SOC_WIFI_CSI_SUPPORT
    Support CSI
SOC_WIFI_MESH_SUPPORT
    Support WIFI MESH
SOC_BLE_SUPPORTED
    Support Bluetooth Low Energy hardware
SOC_BLE_MESH_SUPPORTED
    Support BLE MESH
SOC_BT_CLASSIC_SUPPORTED
    Support Bluetooth Classic hardware

2.10.27 系统时间

概述
ESP32 使用两种硬件时钟源建立和保持系统时间。根据应用目的及对系统时间的精度要求，既可以仅使用其中一种时钟源，也可以同时使用两种时钟源。这两种硬件时钟源为:

- **RTC 定时器**: RTC 定时器在任何睡眠模式下及在任何复位后均可保持系统时间（上电复位除外，因为上电复位会重置 RTC 定时器）。时钟频率偏差取决于 RTC 定时器的时钟源，该偏差只会在睡眠模式下影响时间精度。睡眠模式下，时钟频率为 6.667 μs。
- **高分辨率定时器**: 高分辨率定时器在睡眠模式下及在复位后不可用，但其时间精度更高。该定时器使用 APB_CLK 时钟源（通常为 80 MHz），时钟频率偏移小于 ±10 ppm，时间分辨率为 1 μs。

可供选择的硬件时钟源组合如下所示:

- RTC 和高分辨率定时器（默认）
- RTC
- 高分辨率定时器
- 无
默认时钟源的时间精度最高。建议使用该配置。此外，用户也可以通过配置选项 `CONFIG_NEWLIB_TIME_SYSCALL` 来选择其他时钟源。

**RTC 定时器时钟源**

RTC 定时器有以下时钟源:

- 内置 150 kHz RC 振荡器（默认）：Deep-sleep 模式下电流消耗最低，不依赖任何外部元件。但由于温度波动会影响该时钟源的频率稳定性，在 Deep-sleep 和 Light-sleep 模式下都有可能发生时间偏移。
- 外置 32 kHz 晶振：需要将一个 32 kHz 晶振连接到 32K_XP 和 32K_XN 管脚。频率稳定性更高，但在 Deep-sleep 模式下电流消耗较高（比默认模式高 1 μA）。
- 管脚 32K_XN 外置 32 kHz 振荡器：允许使用外部电路产生的 32 kHz 时钟。外部时钟信号必须连接到管脚 32K_XN。正弦波信号的振幅应小于 1.2 V，方波信号的振幅应小于 1 V。正常模式下，电压范围应为 0.1 < Vcm < 0.5 Vamp，其中 Vamp 代表信号振幅。使用此时钟源时，管脚 32K_XN 无法用作 GPIO 管脚。
- 内置 8.5 MHz 振荡器的 256 分频时钟 (~33 kHz)：频率稳定性优于内置 150 kHz RC 振荡器，同样无需外部元件，但 Deep-sleep 模式下电流消耗更高（比默认模式高 5 μA）。

时钟源的选择取决于系统时间精度要求和睡眠模式下的功耗要求。要修改 RTC 时钟源，请在项目配置中设置 `CONFIG_RTC_CLK_SRC`。

想要了解外置晶振或外置振荡器的更多布线要求，请参考 ESP32 硬件设计指南。

### 获取当前时间

要获取当前时间，请使用 POSIX 函数 `gettimeofday()`。此外，您也可以使用以下标准 C 库函数来获取时间并对它们进行操作：

```c
#include <time.h>

time_t now;
char strftime_buf[64];
struct tm timeinfo;

time(&now);
// 将时区设置为中国标准时间
setenv("TZ", "CST-8", 1);
tzset();
localtime_r(&now, &timeinfo);
strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
ESP_LOGI(TAG, "The current date/time in Shanghai is: %s", strftime_buf);
```

若要求时间的分辨率为 1 μs，请使用以下代码片段:

```c
#include <time.h>

time_t now;
char strftime_buf[64];
struct tm timeinfo;

time(&now);
// 将时区设置为中国标准时间
setenv("TZ", "CST-8", 1);
tzset();
localtime_r(&now, &timeinfo);
strftime(strftime_buf, sizeof(strftime_buf), "%s", &timeinfo);
ESP_LOGI(TAG, "The current date/time in Shanghai is: %s", strftime_buf);
```

若要求时间的分辨率为 1 μs，请使用以下代码片段:
```c
struct timeval tv_now;
gettimeofday(&tv_now, NULL);
int64_t time_us = (int64_t)tv_now.tv_sec * 1000000L + (int64_t)tv_now.tv_usec;
```

### SNTP 时间同步

要设置当前时间，可以使用 POSIX 函数 `settimeofday()` 和 `adjtime()`。 lwIP 中的 SNTP 库会在收到 NTP 服务器的响应报文后，调用这两个函数以更新当前的系统时间。当然，用户可以在 lwIP SNTP 库之外独立地使用这两个函数。

在 lwIP SNTP 库内部调用的函数依赖于系统时间的同步模式。可使用函数 `sntp_set_sync_mode()` 来设置下列同步模式之一。

- **SNTP_SYNC_MODE_IMMED** (默认)：使用函数 `settimeofday()` 后，收到 SNTP 服务器响应时立即更新系统时间。
- **SNTP_SYNC_MODE_SMOOTH**：使用函数 `adjtime()` 后，通过逐渐减小时间误差，平滑地更新时间。如果 SNTP 响应报文中的时间与当前系统时间相差大于 35 分钟，则会通过 `settimeofday()` 立即更新系统时间。

lwIP SNTP 库提供了 API 函数，用于设置某个事件的回调函数。您可能需要使用以下函数：

- `sntp_set_time_sync_notification_cb()`：用于设置回调函数，通知时间同步的过程。
- `sntp_get_sync_status()`：用于获取或设置时间同步状态。

通过 SNTP 开始时间同步，只需调用以下三个函数：

```c
sntp_setoperatingmode(SNTP_OPMODE_POLL);
sntp_setservername(0, "pool.ntp.org");
sntp_init();
```

添加此初始化代码后，应用程序将定期同步时间。时间同步周期由`CONFIG_LWIP_SNTP_UPDATE_DELAY`设置（默认为一小时）。如需修改，请在项目配置中设置`CONFIG_LWIP_SNTP_UPDATE_DELAY`。

如需查看示例代码，请前往 `protocols/sntp` 目录。该目录下的示例展示了如何基于 lwIP SNTP 库实现时间同步。

### 时区

要设置本地时区，请使用以下 POSIX 函数：

1. 调用 `setenv()`，将 `TZ` 环境变量根据设备位置设置为正确的值。时间字符串的格式与 GNU libc 文档中描述的相同（但实现方式不同）。
2. 调用 `tzset()`，为新的时区更新 C 库的运行数据。

完成上述步骤后，请调用标准 C 库函数 `localtime()`。该函数将返回排除时区偏差和夏令时干扰后的准确本地时间。

### 64 位 time_t

ESP-IDF 默认使用 32 位的 `time_t` 类型。为解决 Y2K38 漏洞，您在构建应用程序时可能需要使用 64 位的 `time_t` 类型。

目前，完成这一操作需要从头开始构建交叉编译器工具链，具体步骤请参阅 Linux 和 macOS 平台工具链的标准设置。要在工具链中启用对 64 位 `time_t` 的支持，您需要在构建工具链之前从 `crosstool-NG/samples/xtensa-esp32-elf/crosstool.config` 文件中删除 `--enable-newlib-long-time_t` 选项。

如需使程序同时兼容 32 位和 64 位的 `time_t`，可以使用以下方法：

- 在 C 或 C++ 源文件中，如果 `time_t` 是 32 位的，编译器会预定义 `_USE_LONG_TIME_T` 宏，该宏定义在 `<sys/types.h>` 中。
In CMake files, ESP-IDF configuration attribute `TIME_T_SIZE` will be set to the size of `time_t`. You can use `idf_build_get_property(var TIME_T_SIZE)` to get the value of this attribute and set it in the CMake variable `var`. For more details about `idf_build_get_property` information, see `ESP-IDF CMake API`.

Noting, the size of the `time_t` type will also affect the size of other types, such as `struct timeval`, `struct stat` and `struct utimbuf`.

### API 参考

#### Header File

- `components/lwip/include/apps/esp_sntp.h`

### Functions

- `void sntp_sync_time(struct timeval *tv)`
  - This function updates the system time.
  
  This is a weak-linked function. It is possible to replace all SNTP update functionality by placing a `sntp_sync_time()` function in the app firmware source. If the default implementation is used, calling `sntp_set_sync_mode()` allows the time synchronization mode to be changed to immediate or smooth. If a callback function is registered via `sntp_set_time_sync_notification_cb()`, it will be called following time synchronization.

  参数 `tv` - Time received from SNTP server.

- `void sntp_set_sync_mode(sntp_sync_mode_t sync_mode)`
  - Set the sync mode.
  
  Modes allowed: `SNTP_SYNC_MODE_IMMED` and `SNTP_SYNC_MODE_SMOOTH`.

  参数 `sync_mode` - Sync mode.

- `sntp_sync_mode_t sntp_get_sync_mode(void)`
  - Get set sync mode.

  返回 `SNTP_SYNC_MODE_IMMED`: Update time immediately.
  `SNTP_SYNC_MODE_SMOOTH`: Smooth time updating.

- `sntp_sync_status_t sntp_get_sync_status(void)`
  - Get status of time sync.

  After the update is completed, the status will be returned as `SNTP_SYNC_STATUS_COMPLETED`. After that, the status will be reset to `SNTP_SYNC_STATUS_RESET`. If the update operation is not completed yet, the status will be `SNTP_SYNC_STATUS_RESET`. If a smooth mode was chosen and the synchronization is still continuing (adjtime works), then it will be `SNTP_SYNC_STATUS_IN_PROGRESS`.

  返回 `SNTP_SYNC_STATUS_RESET`: Reset status. `SNTP_SYNC_STATUS_COMPLETED`: Time is synchronized. `SNTP_SYNC_STATUS_IN_PROGRESS`: Smooth time sync in progress.

- `void sntp_set_sync_status(sntp_sync_status_t sync_status)`
  - Set status of time sync.

  参数 `sync_status` - status of time sync (see `sntp_sync_status_t`)

- `void sntp_set_time_sync_notification_cb(sntp_sync_time_cb_t callback)`
  - Set a callback function for time synchronization notification.

  参数 `callback` - a callback function
void \texttt{sntp\_set\_sync\_interval} (uint32\_t interval\_ms)
Set the sync interval of SNTP operation.

Note: SNTPv4 RFC 4330 enforces a minimum sync interval of 15 seconds. This sync interval will be used in the next attempt update time through SNTP. To apply the new sync interval call the \texttt{sntp\_restart()} function, otherwise, it will be applied after the last interval expired.

\textbf{参数} \texttt{interval\_ms} – The sync interval in ms. It cannot be lower than 15 seconds, otherwise 15 seconds will be set.

uint32\_t \texttt{sntp\_get\_sync\_interval} (void)
Get the sync interval of SNTP operation.

\textbf{return} the sync interval

bool \texttt{sntp\_restart} (void)
Restart SNTP.

\textbf{return} True - Restart False - SNTP was not initialized yet

\textbf{Type Definitions}

\textbf{typedef} void (*\texttt{sntp\_sync\_time\_cb\_t})(struct timeval *tv)
SNTP callback function for notifying about time sync event.

\textbf{Param} \texttt{tv} Time received from SNTP server.

\textbf{Enumerations}

\textbf{enum} \texttt{sntp\_sync\_mode\_t}
SNTP time update mode.

\textbf{Values}:

\textbf{enumerator} \texttt{SNTP\_SYNC\_MODE\_IMMED}
Update system time immediately when receiving a response from the SNTP server.

\textbf{enumerator} \texttt{SNTP\_SYNC\_MODE\_SMOOTH}
Smooth time updating. Time error is gradually reduced using \texttt{adjtime} function. If the difference between SNTP response time and system time is large (more than 35 minutes) then update immediately.

\textbf{enum} \texttt{sntp\_sync\_status\_t}
SNTP sync status.

\textbf{Values}:

\textbf{enumerator} \texttt{SNTP\_SYNC\_STATUS\_RESET}

\textbf{enumerator} \texttt{SNTP\_SYNC\_STATUS\_COMPLETED}

\textbf{enumerator} \texttt{SNTP\_SYNC\_STATUS\_IN\_PROGRESS}

2.10.28 The himem allocation API
Overview

The ESP32 can access external SPI RAM transparently, so you can use it as normal memory in your program code. However, because the address space for external memory is limited in size, only the first 4MiB can be used as such. Access to the remaining memory is still possible, however this needs to go through a bankswitching scheme controlled by the himem API.

Specifically, what is implemented by the himem API is a bankswitching scheme. Hardware-wise, the 4MiB region for external SPI RAM is mapped into the CPU address space by a MMU, which maps a configurable 32K bank/page of external SPI RAM into each of the 32K pages in the 4MiB region accessed by the CPU. For external memories that are <=4MiB, this MMU is configured to unity mapping, effectively mapping each CPU address 1-to-1 to the external SPI RAM address.

In order to use the himem API, you have to enable it in the menuconfig using $CONFIG_SPIRAM_BANKSWITCH_ENABLE$, as well as set the amount of banks reserved for this in $CONFIG_SPIRAM_BANKSWITCH_RESERVE$. This decreases the amount of external memory allocated by functions like malloc(), but it allows you to use the himem api to map any of the remaining memory into the reserved banks.

The himem API is more-or-less an abstraction of the bankswitching scheme: it allows you to claim one or more banks of address space (called ‘regions’ in the API) as well as one or more of banks of memory to map into the ranges.

Example

An example doing a simple memory test of the high memory range is available in esp-idf: system/himem

API Reference

Header File

- components/esp_psram/include/esp32/himem.h

Functions

```c
esp_err_t esp_himem_alloc(size_t size, esp_himem_handle_t *handle_out)
```

Allocate a block in high memory.

参数

- **size**: Size of the to-be-allocated block, in bytes. Note that this needs to be a multiple of the external RAM mmu block size (32K).
- **handle_out**: [out] Handle to be returned

返回

- ESP_OK if successful
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_SIZE if size is not a multiple of 32K

```c
esp_err_t esp_himem_alloc_map_range(size_t size, esp_himem_rangehandle_t *handle_out)
```

Allocate a memory region to map blocks into.

参数

- **size**: Size of the range to be allocated. Note this needs to be a multiple of the external RAM mmu block size (32K).
- **handle_out**: [out] Handle to be returned

返回

- ESP_OK if successful
- ESP_ERR_NO_MEM if out of memory or address space
- ESP_ERR_INVALID_SIZE if size is not a multiple of 32K

```c
esp_err_t esp_himem_map(esp_himem_handle_t handle, esp_himem_rangehandle_t range, size_t ram_offset, size_t range_offset, size_t len, int flags, void **out_ptr)
```

Map a block of high memory into the CPUs address space.

This effectively makes the block available for read/write operations.
Chapter 2. API 参考

备注：The region to be mapped needs to have offsets and sizes that are aligned to the SPI RAM MMU block size (32K)

参数

- **handle** - Handle to the block of memory, as given by `esp_himem_alloc`
- **range** - Range handle to map the memory in
- **ram_offset** - Offset into the block of physical memory of the block to map
- **range_offset** - Offset into the address range where the block will be mapped
- **len** - Length of region to map
- **flags** - One of `ESP_HIMEM_MAPFLAG_*`
- **out_ptr** - [out] Pointer to variable to store resulting memory pointer in

返回 - ESP_OK if the memory could be mapped

- ESP_ERR_INVALID_ARG if offset, range or len aren’t MMU-block-aligned (32K)
- ESP_ERR_INVALID_SIZE if the offsets/lengths don’t fit in the allocated memory or range
- ESP_ERR_INVALID_STATE if a block in the selected ram offset/length is already mapped, or if a block in the selected range offset/length already has a mapping.

```c
esp_err_t esp_himem_free(esp_himem_handle_t handle)
```

Free a block of physical memory.

This clears out the associated handle making the memory available for re-allocation again. This will only succeed if none of the memory blocks currently have a mapping.

参数 **handle** - Handle to the block of memory, as given by `esp_himem_alloc`

返回 - ESP_OK if the memory is succesfully freed

- ESP_ERR_INVALID_ARG if the handle still is (partially) mapped

```c
esp_err_t esp_himem_free_map_range(esp_himem_rangehandle_t handle)
```

Free a mapping range.

This clears out the associated handle making the range available for re-allocation again. This will only succeed if none of the range blocks currently are used for a mapping.

参数 **handle** - Handle to the range block, as given by `esp_himem_alloc_map_range`

返回 - ESP_OK if the memory is succesfully freed

- ESP_ERR_INVALID_ARG if the handle still is (partially) mapped to

```c
esp_err_t esp_himem_unmap(esp_himem_rangehandle_t range, void *ptr, size_t len)
```

Unmap a region.

参数

- **range** - Range handle
- **ptr** - Pointer returned by `esp_himem_map`
- **len** - Length of the block to be unmapped. Must be aligned to the SPI RAM MMU blocksize (32K)

返回 - ESP_OK if the memory is succesfully unmapped,

- ESP_ERR_INVALID_ARG if ptr or len are invalid.

```c
size_t esp_himem_get_phys_size(void)
```

Get total amount of memory under control of himem API.

返回 Amount of memory, in bytes

```c
size_t esp_himem_get_free_size(void)
```

Get free amount of memory under control of himem API.

返回 Amount of free memory, in bytes
size_t esp_himem_reserved_area_size (void)
Get amount of SPI memory address space needed for bankswitching.

备注：This is also weakly defined in esp32/spiram.c and returns 0 there, so if no other function in this file is
used, no memory is reserved.

返回 Amount of reserved area, in bytes

 Macros

ESP_HIMEM_BLKSZ

ESP_HIMEM_MAPFLAG_RO
Indicates that a mapping will only be read from. Note that this is unused for now.

 Type Definitions

typedef struct esp_himem_ramdata_t *esp_himem_handle_t
typedef struct esp_himem_rangedata_t *esp_himem_rangehandle_t

2.10.29 ULP 协处理器编程

ULP（Ultra Low Power，超低功耗）协处理器是一种简单的有限状态机（FSM），可以在主处理器处于
深度睡眠模式时，使用 ADC、温度传感器和外部 I2C 传感器执行测量操作。ULP 协处理器可以访问
RTC_SLOW_MEM 内存区域及 RTC_CNTL、RTC_IO、SARADC 外设中的寄存器。ULP 协处理器使用 32
位固定宽度的指令，32 位内存寻址，配备 4 个 16 位通用寄存器。在 ESP-IDF 项目中，此协处理器称作
ULP FSM。

安装工具链

ULP FSM 协处理器代码由汇编语言编写，使用 binutils-esp32ulp 工具链 进行编译。
如果您已经按照 快速入门指南 中的介绍安装好了 ESP-IDF 及其 CMake 构建系统，那么 ULP 工具链已经
被默认安装到了您的开发环境中。

编写 ULP FSM

使用受支持的指令集即可编写 ULP FSM 协处理器，此外也可使用主处理器上的 C 语言宏进行编程。以下小节分别介绍了这两种方法：

ESP32 ULP coprocessor instruction set  This document provides details about the instructions used by ESP32
ULP FSM coprocessor assembler.

ULP FSM coprocessor has 4 16-bit general purpose registers, labeled R0, R1, R2, R3. It also has an 8-bit counter
register (stage_cnt) which can be used to implement loops. Stage count register is accessed using special instructions.
ULP coprocessor can access 8k bytes of RTC_SLOW_MEM memory region. Memory is addressed in 32-bit word
units. It can also access peripheral registers in RTC_CNTL, RTC_IO, and SENS peripherals.
All instructions are 32-bit. Jump instructions, ALU instructions, peripheral register and memory access instructions are executed in 1 cycle. Instructions which work with peripherals (TSENS, ADC, I2C) take variable number of cycles, depending on peripheral operation.

The instruction syntax is case insensitive. Upper and lower case letters can be used and intermixed arbitrarily. This is true both for register names and instruction names.

**Note about addressing**    ESP32 ULP FSM coprocessor’s JUMP, ST, LD family of instructions expect the address argument to be expressed in the following way depending on the type of address argument used:

- When the address argument is presented as a label then the instruction expects the address to be expressed as 32-bit words.

  Consider the following example program:

  ```
  entry:
  NOP
  NOP
  NOP
  NOP
  loop:
  MOVE R1, loop
  JUMP R1
  ```

  When this program is assembled and linked, address of label `loop` will be equal to 16 (expressed in bytes). However, `JUMP` instruction expects the address stored in register `R1` to be expressed in 32-bit words. To account for this common use case, the assembler will convert the address of label `loop` from bytes to words, when generating the `MOVE` instruction. Hence, the code generated code will be equivalent to:

  ```
  0000  NOP
  0004  NOP
  0008  NOP
  000c  NOP
  0010  MOVE R1, 4
  0014  JUMP R1
  ```

- The other case is when the argument of MOVE instruction is not a label but a constant. In this case assembler will **use the value as is**, without any conversion:

  ```
  .set    val, 0x10
  MOVE    R1, val
  ```

  In this case, value loaded into `R1` will be `0x10`.

  However, when an immediate value is used as an offset in LD and ST instructions, the assembler considers the address argument in bytes and converts it to 32-bit words before executing the instruction:

  ```
  ST R1, R2, 4 // offset = 4 bytes; Mem[R2 + 4 / 4] = R1
  ```

  In this case, the value in `R1` is stored at the memory location pointed by `[R2 + offset / 4]`.

  Consider the following code:

  ```
  .global array
  array: .long 0
  .long 0
  .long 0
  .long 0
  .long 0
  MOVE R1, array
  MOVE R2, 0x1234
  ST R2, R1, 0 // write value of R2 into the first array element, i.e. array[0]
  ST R2, R1, 4 // write value of R2 into the second array element (4 byte offset), i.e. array[1]
  ```
### ADD

**Syntax**  
ADD \( R_{dst}, R_{src1}, R_{src2} \)

**Operands**  
- \( R_{dst} \) - Register \([0..3]\)
- \( R_{src1} \) - Register \([0..3]\)
- \( R_{src2} \) - Register \([0..3]\)
- \( \text{Imm} \) - 16-bit signed value

**Cycles**  
2 cycles to execute, 4 cycles to fetch next instruction

**Description**  
The instruction adds source register to another source register or to a 16-bit signed value and stores the result in the destination register.

**Examples**

1. `ADD R1, R2, R3`  // \( R1 = R2 + R3 \)
2. `ADD R1, R2, 0x1234`  // \( R1 = R2 + 0x1234 \)

---

**Note about instruction execution time**  
ULP coprocessor is clocked from RTC_FAST_CLK, which is normally derived from the internal 8MHz oscillator. Applications which need to know exact ULP clock frequency can calibrate it against the main XTAL clock:

```c
#include "soc/rtc.h"

// calibrate 8M/256 clock against XTAL, get 8M/256 clock period
uint32_t rtc_8md256_period = rtc_clk_cal(RTC_CAL_8MD256, 100);
uint32_t rtc_fast_freq_hz = 1000000ULL * (1 << RTC_CLK_CAL_FRACT) * 256 / rtc_8md256_period;
```

ULP coprocessor needs certain number of clock cycles to fetch each instruction, plus certain number of cycles to execute it, depending on the instruction. See description of each instruction below for details on the execution time.

Instruction fetch time is:
- 2 clock cycles — for instructions following ALU and branch instructions.
- 4 clock cycles — in other cases.

Note that when accessing RTC memories and RTC registers, ULP coprocessor has lower priority than the main CPUs. This means that ULP coprocessor execution may be suspended while the main CPUs access same memory region as the ULP.

The detailed description of all instructions is presented below:

**NOP - no operation**

**Syntax** NOP

**Operands** None

**Cycles** 2 cycle to execute, 4 cycle to fetch next instruction

**Description** No operation is performed. Only the PC is incremented.

**Example**

1: NOP
SUB - Subtract from register

Syntax  SUB Rdst, Rsrc1, Rsrc2
       SUB Rdst, Rsrc1, imm

Operands
- Rdst - Register R[0..3]
- Rsrc1 - Register R[0..3]
- Rsrc2 - Register R[0..3]
- Imm - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction subtracts the source register from another source register or subtracts a 16-bit signed value from a source register, and stores the result to the destination register.

Examples:
1: SUB R1, R2, R3  // R1 = R2 - R3
2: sub R1, R2, 0x1234 // R1 = R2 - 0x1234
3: .set value1, 0x03  // constant value1=0x03
   SUB R1, R2, value1  // R1 = R2 - value1
4: .global label  // declaration of variable label
   add R1, R2, label  // R1 = R2 + label
   ...
   label: nop  // definition of variable label

AND - Bitwise logical AND of two operands

Syntax  AND Rdst, Rsrc1, Rsrc2
       AND Rdst, Rsrc1, imm

Operands
- Rdst - Register R[0..3]
- Rsrc1 - Register R[0..3]
- Rsrc2 - Register R[0..3]
- Imm - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does a bitwise logical AND of a source register and another source register or a 16-bit signed value and stores the result to the destination register.

Examples:
1: AND R1, R2, R3  // R1 = R2 & R3
2: AND R1, R2, 0x1234 // R1 = R2 & 0x1234
3: .set value1, 0x03  // constant value1=0x03
   AND R1, R2, value1  // R1 = R2 & value1
4: .global label  // declaration of variable label
Chapter 2. API

AND R1, R2, label   // R1 = R2 & label
...                  
label: nop           // definition of variable label

OR - Bitwise logical OR of two operands

Syntax  OR Rdst, Rsrc1, Rsrc2
       OR Rdst, Rsrc1, imm

Operands
  • Rdst - Register R[0..3]
  • Rsrc1 - Register R[0..3]
  • Rsrc2 - Register R[0..3]
  • Imm - 16-bit signed value

Cycles  2 cycles to execute, 4 cycles to fetch next instruction

Description  The instruction does a bitwise logical OR of a source register and another source register or a 16-bit signed value and stores the result to the destination register.

Examples:
1:   OR R1, R2, R3   // R1 = R2 || R3
2:   OR R1, R2, 0x1234 // R1 = R2 || 0x1234
3:   .set value1, 0x03 // constant value1=0x03
    OR R1, R2, value1   // R1 = R2 || value1
4:   .global label    // declaration of variable label
    OR R1, R2, label    // R1 = R2 || label
    ...                
    label: nop         // definition of variable label

LSH - Logical Shift Left

Syntax  LSH Rdst, Rsrc1, Rsrc2
       LSH Rdst, Rsrc1, imm

Operands
  • Rdst - Register R[0..3]
  • Rsrc1 - Register R[0..3]
  • Rsrc2 - Register R[0..3]
  • Imm - 16-bit signed value

Cycles  2 cycles to execute, 4 cycles to fetch next instruction

Description  The instruction does a logical shift to left of the source register by the number of bits from another source register or a 16-bit signed value and stores the result to the destination register.

备注: Shift operations which are greater than 15 bits will have an undefined result.

Examples:
1:   LSH R1, R2, R3   // R1 = R2 << R3
2:   LSH R1, R2, 0x03 // R1 = R2 << 0x03
3:   .set value1, 0x03 // constant value1=0x03
    LSH R1, R2, value1   // R1 = R2 << value1
4:   .global label    // declaration of variable label
**Chapter 2. API 参考**

---

**LSH**

**R1, R2, label**  // R1 = R2 << label

---

**RSH - Logical Shift Right**

**Syntax**

RSH  

**Rdst, Rsrc1, Rsrc2**

RSH  

**Rdst, Rsrc1, imm**

**Operands**

- **Rdst** - Register R[0..3]
- **Rsrc1** - Register R[0..3]
- **Rsrc2** - Register R[0..3]
- **Imm** - 16-bit signed value

**Cycles**

2 cycles to execute, 4 cycles to fetch next instruction

**Description**

The instruction does a logical shift to right of a source register by the number of bits from another source register or a 16-bit signed value and stores the result to the destination register.

**Examples:**

1: RSH R1, R2, R3  // R1 = R2 >> R3

2: RSH R1, R2, 0x03  // R1 = R2 >> 0x03

3: .set value1, 0x03  // constant value1=0x03
   RSH R1, R2, value1  // R1 = R2 >> value1

4: .global label  // declaration of variable label
   RSH R1, R2, label  // R1 = R2 >> label
   label: nop  // definition of variable label

---

**MOVE - Move to register**

**Syntax**

MOVE  

**Rdst, Rsrc**

MOVE  

**Rdst, imm**

**Operands**

- **Rdst** – Register R[0..3]
- **Rsrc** – Register R[0..3]
- **Imm** – 16-bit signed value

**Cycles**

2 cycles to execute, 4 cycles to fetch next instruction

**Description**

The instruction moves the value from the source register or a 16-bit signed value to the destination register.

**Examples:**

1: MOVE R1, R2  // R1 = R2

2: MOVE R1, 0x03  // R1 = 0x03

3: .set value1, 0x03  // constant value1=0x03
   MOVE R1, value1  // R1 = value1

4: .global label  // declaration of label
   MOVE R1, label  // R1 = address_of(label) / 4

---

Espressif Systems 2071  Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
Chapter 2. API

### ST – Store data to the memory

**Syntax** ST `Rsrc, Rdst, offset`

**Operands**
- `Rsrc` – Register R[0..3], holds the 16-bit value to store
- `Rdst` – Register R[0..3], address of the destination, in 32-bit words
- `Offset` – 13-bit signed value, offset in bytes

**Cycles** 4 cycles to execute, 4 cycles to fetch next instruction

**Description** The instruction stores the 16-bit value of `Rsrc` to the lower half-word of memory with address `Rdst+offset`. The upper half-word is written with the current program counter (PC) (expressed in words, shifted left by 5 bits) OR’d with `Rdst` (0..3):

```
Mem[Rdst + offset / 4][31:0] = (PC[10:0], 3'b0, Rdst, Rsrc[15:0])
```

The application can use the higher 16 bits to determine which instruction in the ULP program has written any particular word into memory.

### Notes about addressing

Note: The offset specified in bytes is converted to a 32-bit word offset before execution. See the section about addressing for more details.

### Examples:

1:  
```
ST R1, R2, 0x12 // MEM[R2 + 0x12 / 4] = R1
```

2:  
```
.data  
Addr1: .word 123  // Define label Addr1 16 bit
.set offs, 0x00  // Define constant offs
.text  
MOVE R1, 1  // R1 = 1
MOVE R2, Addr1  // R2 = Addr1
ST R1, R2, offs // MEM[R2 + 0 / 4] = R1  
// MEM[Addr1 + 0] will be 32'h600001
```

### LD – Load data from the memory

**Syntax** LD `Rdst, Rsrc, offset`

**Operands**
- `Rdst` – Register R[0..3], destination
- `Rsrc` – Register R[0..3], holds address of destination, in 32-bit words
- `Offset` – 13-bit signed value, offset in bytes

**Cycles** 4 cycles to execute, 4 cycles to fetch next instruction

**Description** The instruction loads the lower 16-bit half-word from memory with address `[Rsrc + offset / 4]` into the destination register `Rdst`:

```
Rdst[15:0] = Mem[Rsrc + offset / 4][15:0]
```

### Notes about addressing

Note: The offset specified in bytes is converted to a 32-bit word offset before execution. See the section about addressing for more details.

### Examples:
Chapter 2. API 参考

1:  LD  R1, R2, 0x12  // R1 = MEM[R2 + 0x12 / 4]

2:
.data
 Addr1: .word 123  // Data section definition
 .set offs, 0x00  // Define constant offs
 .text
  MOVE R1, 1  // R1 = 1
  MOVE R2, Addr1  // R2 = Addr1 / 4 (address of label is
                 // converted into words)
  LD  R1, R2, offs  // R1 = MEM[R2 + 0]
                 // R1 will be 123

JUMP – Jump to an absolute address

Syntax  JUMP Rdst
       JUMP ImmAddr
       JUMP Rdst, Condition
       JUMP ImmAddr, Condition

Operands
  • Rdst – Register R[0..3] containing address to jump to (expressed in 32-bit words)
  • ImmAddr – 13 bits address (expressed in bytes), aligned to 4 bytes
  • Condition:
    – EQ – jump if last ALU operation result was zero
    – OV – jump if last ALU has set overflow flag

Cycles  2 cycles to execute, 2 cycles to fetch next instruction

Description The instruction makes jump to the specified address. Jump can be either unconditional or based on an
ALU flag.

Examples:

1:  JUMP  R1  // Jump to address in R1 (address in R1 is in
         ←32-bit words)

2:  JUMP  0x120, EQ // Jump to address 0x120 (in bytes) if ALU
         ←result is zero

3:  JUMP  label  // Jump to label
     ...
     label: nop  // Definition of label

4:  .global  label  // Declaration of global label
  MOVE  R1, label  // R1 = label (value loaded into R1 is in words)
  JUMP  R1  // Jump to label
     ...
     label: nop  // Definition of label

JUMPR – Jump to a relative offset (condition based on R0)

Syntax  JUMPR Step, Threshold, Condition

Operands
  • Step – relative shift from current position, in bytes
  • Threshold – threshold value for branch condition
  • Condition:
    – EQ (equal) – jump if value in R0 == threshold
    – LT (less than) – jump if value in R0 < threshold
    – LE (less or equal) – jump if value in R0 <= threshold
    – GT (greater than) – jump if value in R0 > threshold
    – GE (greater or equal) – jump if value in R0 >= threshold
Cycles  Conditions $EQ$, $GT$ and $LT$: 2 cycles to execute, 2 cycles to fetch next instruction Conditions $LE$ and $GE$ are implemented in the assembler using two JUMPR instructions:

```c
// JUMPR target, threshold, LE is implemented as:
JUMPR target, threshold, EQ
JUMPR target, threshold, LT

// JUMPR target, threshold, GE is implemented as:
JUMPR target, threshold, EQ
JUMPR target, threshold, GT
```

Therefore the execution time will depend on the branches taken: either 2 cycles to execute + 2 cycles to fetch, or 4 cycles to execute + 4 cycles to fetch.

**Description** The instruction makes a jump to a relative address if condition is true. Condition is the result of comparison of R0 register value and the threshold value.

**Examples:**

```c
1: pos: JUMPR 16, 20, GE  // Jump to address (position + 16 bytes) if value in R0 >= 20

2:       // Down counting loop using R0 register
         MOVE R0, 16    // load 16 into R0
         label: SUB R0, R0, 1   // R0--
         NOP     // do something
         JUMPR label, 1, GE // jump to label if R0 >= 1
```

**JUMPS**—Jump to a relative address (condition based on stage count)

**Syntax** JUMPS Step, Threshold, Condition

**Operands**

- **Step**—relative shift from current position, in bytes
- **Threshold**—threshold value for branch condition
- **Condition:**
  - $EQ$ (equal)—jump if value in stage_cnt == threshold
  - $LT$ (less than)—jump if value in stage_cnt < threshold
  - $LE$ (less or equal)—jump if value in stage_cnt <= threshold
  - $GT$ (greater than)—jump if value in stage_cnt > threshold
  - $GE$ (greater or equal)—jump if value in stage_cnt >= threshold

**Cycles** 2 cycles to execute, 2 cycles to fetch next instruction:

```c
// JUMPS target, threshold, EQ is implemented as:
JUMPS next, threshold, LT
JUMPS target, threshold, LE

// JUMPS target, threshold, GT is implemented as:
JUMPS next, threshold, LE
JUMPS target, threshold, GE
```

Therefore the execution time will depend on the branches taken: either 2 cycles to execute + 2 cycles to fetch, or 4 cycles to execute + 4 cycles to fetch.

**Description** The instruction makes a jump to a relative address if condition is true. Condition is the result of comparison of count register value and threshold value.

**Examples:**
1: pos: JUMPS 16, 20, EQ // Jump to (position + 16 bytes) if stage_cnt_ -= 20

2: // Up counting loop using stage count register
STAGE_RST // set stage_cnt to 0
label: STAGE_INC 1 // stage_cnt++
NOP // do something
JUMPS label, 16, LT // jump to label if stage_cnt < 16

**STAGE_RST** – Reset stage count register

**Syntax** STAGE_RST

**Operands** No operands

**Description** The instruction sets the stage count register to 0

**Cycles** 2 cycles to execute, 4 cycles to fetch next instruction

**Examples:**

1: STAGE_RST // Reset stage count register

**STAGE_INC** – Increment stage count register

**Syntax** STAGE_INC Value

**Operands**

- Value – 8 bits value

**Cycles** 2 cycles to execute, 4 cycles to fetch next instruction

**Description** The instruction increments the stage count register by the given value.

**Examples:**

1: STAGE_INC 10 // stage_cnt += 10

2: // Up counting loop example:
STAGE_RST // set stage_cnt to 0
label: STAGE_INC 1 // stage_cnt++
NOP // do something
JUMPS label, 16, LT // jump to label if stage_cnt < 16

**STAGE_DEC** – Decrement stage count register

**Syntax** STAGE_DEC Value

**Operands**

- Value – 8 bits value

**Cycles** 2 cycles to execute, 4 cycles to fetch next instruction

**Description** The instruction decrements the stage count register by the given value.

**Examples:**

1: STAGE_DEC 10 // stage_cnt -= 10;

2: // Down counting loop example
STAGE_RST // set stage_cnt to 0
STAGE_INC 16 // increment stage_cnt to 16
label: STAGE_DEC 1 // stage_cnt--;
NOP // do something
JUMPS label, 0, GT // jump to label if stage_cnt > 0
Chapter 2. API 参考

HALT – End the program

Syntax

HALT

Operands

No operands

Cycles

2 cycles to execute

Description

The instruction halts the ULP coprocessor and restarts the ULP wake up timer, if it is enabled.

Examples:

1: HALT // Halt the coprocessor

WAKE – Wake up the chip

Syntax

WAKE

Operands

No operands

Cycles

2 cycles to execute, 4 cycles to fetch next instruction

Description

The instruction sends an interrupt from the ULP coprocessor to the RTC controller.

• If the SoC is in deep sleep mode, and ULP wake up is enabled, this causes the SoC to wake up.
• If the SoC is not in deep sleep mode, and ULP interrupt bit (RTC_CNTL_ULP_CP_INT_ENA) is set in RTC_CNTL_INT_ENA_REG register, RTC interrupt will be triggered.

Examples:

1: is_rdy_for_wakeup: // Read RTC_CNTL_RDY_FOR_WAKEUP bit
    READ_RTC_FIELD(RTC_CNTL_LOW_POWER_ST_REG, RTC_CNTL_RDY_FOR_WAKEUP)
    AND r0, r0, 1
    JUMP is_rdy_for_wakeup, eq // Retry until the bit is set
    WAKE // Trigger wake up
    REG_WR 0x006, 24, 24, 0 // Stop ULP timer (clear RTC_CNTL_ULP_CP_
←SLP_TIMER_EN)
    HALT // Stop the ULP program
    // After these instructions, SoC will wake up,
    // and ULP will not run again until started by the main program.

1: check_wakeup: // Read RTC_CNTL_RDY_FOR_WAKEUP and RTC_
←CNTL_MAIN_STATE_IN_IDLE bit
    READ_RTC_REG(RTC_CNTL_LOW_POWER_ST_REG, 27, 1)
    MOVE r1, r0 // Copy result in to r1
    READ_RTC_FIELD(RTC_CNTL_LOW_POWER_ST_REG, RTC_CNTL_RDY_FOR_WAKEUP)
    OR r0, r0, r1
    JUMP check_wakeup, eq // Retry until either of the bit are set
    WAKE // Trigger wake up
    HALT // Stop the ULP program

SLEEP – set ULP wake up timer period

Syntax

SLEEP sleep_reg

Operands

• sleep_reg – 0..4, selects one of SENS_ULP_CP_SLEEP_CYCx_REG registers.

Cycles

2 cycles to execute, 4 cycles to fetch next instruction
**Description** The instruction selects which of the `SENS_ULP_CP_SLEEP_CYCx_REG (x = 0..4)` register values is to be used by the ULP wakeup timer as wakeup period. By default, the value from `SENS_ULP_CP_SLEEP_CYC0_REG` is used.

**Examples:**

1:    SLEEP 1 // Use period set in SENS_ULP_CP_SLEEP_CYC1_REG
2:    .set sleep_reg, 4 // Set constant
       SLEEP sleep_reg // Use period set in SENS_ULP_CP_SLEEP_CYC4_REG

**WAIT --wait some number of cycles**

**Syntax** WAIT Cycles

**Operands**

```
• Cycles --number of cycles for wait
```

**Cycles** 2 + Cycles to execute, 4 cycles to fetch next instruction

**Description** The instruction delays for given number of cycles.

**Examples:**

1:    WAIT 10 // Do nothing for 10 cycles
2:    .set wait_cnt, 10 // Set a constant
       WAIT wait_cnt // wait for 10 cycles

**ADC --do measurement with ADC**

**Syntax**

```
• ADC Rdst, Sar_sel, Mux
• ADC Rdst, Sar_sel, Mux, 0 --deprecated form
```

**Operands**

```
• Rdst --Destination Register R[0..3], result will be stored to this register
• Sar_sel --Select ADC: 0 = SARADC1, 1 = SARADC2
• Mux - Enable ADC channel. Channel number is [Mux-1]. If the user passes Mux value 1, then ADC channel 0 gets used.
```

**Cycles** 23 + max(1, SAR_AMP_WAIT1) + max(1, SAR_AMP_WAIT2) + max(1, SAR_AMP_WAIT3) + SARx_SAMPLE_CYCLE + SARx_SAMPLE_BIT cycles to execute, 4 cycles to fetch next instruction

**Description** The instruction makes measurements from ADC.

**Examples:**

```
.. only:: esp32

1: ADC R1, 0, 1 // Measure value using ADC1 channel 0 and store result into R1
```

**I2C_RD - read single byte from I2C slave**

**Syntax**

```
• I2C_RD Sub_addr, High, Low, Slave_sel
```

**Operands**

```
• Sub_addr --Address within the I2C slave to read.
• High, Low --Define range of bits to read. Bits outside of [High, Low] range are masked.
• Slave_sel - Index of I2C slave address to use.
```

**Cycles** Execution time mostly depends on I2C communication time. 4 cycles to fetch next instruction.

**Description** I2C_RD instruction reads one byte from I2C slave with index Slave_sel. Slave address (in 7-bit format) has to be set in advance into `SENS_I2C_SLAVE_ADDRx` register field, where x == Slave_sel. 8 bits of read result is stored into R0 register.
Examples:

I2C_RD - read byte from sub-address

```
1: I2C_RD 0x10, 7, 0, 0 // Read byte from sub-address 0x10 of...
```

```
←slave with address set in SENS_I2C_SLAVE_ADDR0
```

I2C_WR - write single byte to I2C slave

Syntax

```
I2C_WR Sub_addr, Value, High, Low, Slave_sel
```

Operands

```
• Sub_addr – Address within the I2C slave to write.
• Value – 8-bit value to be written.
• High, Low – Define range of bits to write. Bits outside of [High, Low] range are masked.
• Slave_sel - Index of I2C slave address to use.
```

Cycles Execution time mostly depends on I2C communication time. 4 cycles to fetch next instruction.

Description I2C_WR instruction writes one byte to I2C slave with index Slave_sel. Slave address (in 7-bit format) has to be set in advance into SENS_I2C_SLAVE_ADDRx register field, where x == Slave_sel.

Examples:

```
1: I2C_WR 0x20, 0x33, 7, 0, 1 // Write byte 0x33 to sub-address...
←0x20 of slave with address set in SENS_I2C_SLAVE_ADDR1.
```

REG_RD – read from peripheral register

Syntax

```
REG_RD Addr, High, Low
```

Operands

```
• Addr – Register address, in 32-bit words
• High – Register end bit number
• Low – Register start bit number
```

Cycles 4 cycles to execute, 4 cycles to fetch next instruction.

Description The instruction reads up to 16 bits from a peripheral register into a general purpose register: \[ R0 = \text{REG}[\text{Addr}][\text{High:Low}] \].

This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the register, as seen from the ULP, can be calculated from the address of the same register on the DPORT bus as follows:

```
addr_ulp = (addr_dport - DR_REG_RTC_CNTL_BASE) / 4
```

Examples:

```
1: REG_RD 0x120, 7, 4 // load 4 bits: R0 = {12'b0, REG[0x120][7:4]}
```

REG_WR – write to peripheral register

Syntax

```
REG_WR Addr, High, Low, Data
```

Operands

```
• Addr – Register address, in 32-bit words.
• High – Register end bit number
• Low – Register start bit number
• Data – Value to write, 8 bits
```

Cycles 8 cycles to execute, 4 cycles to fetch next instruction.

Description The instruction writes up to 8 bits from an immediate data value into a peripheral register: \[ \text{REG}[\text{Addr}][\text{High:Low}] = \text{data} \].

This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the register, as seen from the ULP, can be calculated from the address of the same register on the DPORT bus as follows:
addr_ulp = (addr_dport - DR_REG_RTCCNTL_BASE) / 4

Examples:

1: REG_WR 0x120, 7, 0, 0x10 // set 8 bits: REG[0x120][7:0] = 0x10

Convenience macros for peripheral registers access

ULP source files are passed through C preprocessor before the assembler. This allows certain macros to be used to facilitate access to peripheral registers.

Some existing macros are defined in `soc/soc_ulp.h` header file. These macros allow access to the fields of peripheral registers by their names. Peripheral registers names which can be used with these macros are the ones defined in `soc/rtc_cntl_reg.h`, `soc/rtc_io_reg.h`, `soc/sens_reg.h`, and `soc/rtc_i2c_reg.h`.

**READ_RTC_REG**(rtc_reg, low_bit, bit_width) Read up to 16 bits from rtc_reg[low_bit + bit_width - 1 : low_bit] into R0. For example:

```c
#include "soc/soc_ulp.h"
#include "soc/rtc_cntl_reg.h"

/* Read 16 lower bits of RTC_CNTL_TIME0_REG into R0 */
READ_RTC_REG(RTC_CNTL_TIME0_REG, 0, 16)
```

**READ_RTC_FIELD**(rtc_reg, field) Read from a field in rtc_reg into R0, up to 16 bits. For example:

```c
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"

/* Read 8-bit SENS_TSENS_OUT field of SENS_SAR_SLAVE_ADDR3_REG into R0 */
READ_RTC_FIELD(SENS_SAR_SLAVE_ADDR3_REG, SENS_TSENS_OUT)
```

**WRITE_RTC_REG**(rtc_reg, low_bit, bit_width, value) Write immediate value into rtc_reg[low_bit + bit_width - 1 : low_bit], bit_width <= 8. For example:

```c
#include "soc/soc_ulp.h"
#include "soc/rtc_io_reg.h"

/* Set BIT(2) of RTC_GPIO_OUT_DATA_W1TS field in RTC_GPIO_OUT_W1TS_REG */
WRITE_RTC_REG(RTC_GPIO_OUT_W1TS_REG, RTC_GPIO_OUT_DATA_W1TS_S + 2, 1, 1)
```

**WRITE_RTC_FIELD**(rtc_reg, field, value) Write immediate value into a field in rtc_reg, up to 8 bits. For example:

```c
#include "soc/soc_ulp.h"
#include "soc/rtc_cntl_reg.h"

/* Set RTC_CNTL_ULP_CP_SLP_TIMER_EN field of RTC_CNTL_STATE0_REG to 0 */
WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
```

Programming ULP FSM coprocessor using C macros (legacy)

In addition to the existing binutils port for the ESP32 ULP coprocessor, it is possible to generate programs for the ULP FSM coprocessor by embedding assembly-like macros into an ESP32 application. Here is an example how this can be done:

```c
const ulp_insn_t program[] = {
    I_MOVI(R3, 16),  // R3 <- 16
    I_LD(R0, R3, 0),  // R0 <- RTC_SLOW_MEM[R3 + 0]
    I_LD(R1, R3, 1),  // R1 <- RTC_SLOW_MEM[R3 + 1]
    I_ADDR(R2, R0, R1),  // R2 <- R0 + R1
    I_ST(R2, R3, 2),  // R2 -> RTC_SLOW_MEM[R2 + 2]
    I_HALT()  // (下頁續接)
};
```

Espressif Systems 2079 Release v5.1-dev-2066-g7869f4e151
The program array is an array of ulp_insn_t, i.e. ULP coprocessor instructions. Each I_XXX preprocessor define translates into a single 32-bit instruction. Arguments of these preprocessor defines can be register numbers (R0 —R3) and literal constants. See the API reference section at the end of this guide for descriptions of instructions and arguments they take.

备注: Because some of the instruction macros expand to inline function calls, defining such array in global scope will cause the compiler to produce an “initializer element is not constant” error. To fix this error, move the definition of instructions array into local scope.

备注: Load, store and move instructions use addresses expressed in 32-bit words. Address 0 corresponds to the first word of RTC_SLOW_MEM. This is different to how address arguments are handled in assembly code of the same instructions. See the section Note about addressing for more details for reference.

To generate branch instructions, special M_ preprocessor defines are used. M_LABEL define can be used to define a branch target. Label identifier is a 16-bit integer. M_Bxxx defines can be used to generate branch instructions with target set to a particular label.

Implementation note: these M_ preprocessor defines will be translated into two ulp_insn_t values: one is a token value which contains label number, and the other is the actual instruction. ulp_process_macros_and_load function resolves the label number to the address, modifies the branch instruction to use the correct address, and removes the extra ulp_insn_t token which contains the label numer.

Here is an example of using labels and branches:

```c
const ulp_insn_t program[] = {
    I_MOVI(R0, 34),        // R0 <- 34
    M_LABEL(1),            // label_1
    I_MOVI(R1, 32),        // R1 <- 32
    I_LD(R1, R1, 0),       // R1 <- RTC_SLOW_MEM[R1]
    I_MOVI(R2, 33),        // R2 <- 33
    I_LD(R2, R2, 0),       // R2 <- RTC_SLOW_MEM[R2]
    I_SUBR(R3, R1, R2),    // R3 <- R1 - R2
    I_ST(R3, R0, 0),       // R3 -> RTC_SLOW_MEM[R0 + 0]
    I_ADDI(R0, R0, 1),     // R0++
    M_BL(1, 64),           // if (R0 < 64) goto label_1
    I_HALT(),
};
RTC_SLOW_MEM[32] = 42;
RTC_SLOW_MEM[33] = 18;
size_t load_addr = 0;
size_t size = sizeof(program)/sizeof(ulp_insn_t);
ulp_process_macros_and_load(load_addr, program, &size);
ulp_run(load_addr);
```

API Reference

Header File

- components/ulp/ulp_fsm/include/esp32/ulp.h
Functions
static inline uint32_t SOC_REG_TO_ULP_PERIPH_SEL (uint32_t reg)
Map SoC peripheral register to periph_sel field of RD_REG and WR_REG instructions.
参数 reg – peripheral register in RTC_CNTL_, RTC_IO_, SENS_, RTC_I2C peripherals.
返回 periph_sel value for the peripheral to which this register belongs.

Unions
union ulp_insn
#include <ulp.h> Instruction format structure.
All ULP instructions are 32 bit long. This union contains field layouts used by all of the supported instructions.
This union also includes a special “macro” instruction layout. This is not a real instruction which can be executed
by the CPU. It acts as a token which is removed from the program by the ulp_process_macros_and_load
function.
These structures are not intended to be used directly. Preprocessor definitions provided below fill the fields of
these structure with the right arguments.

Public Members

uint32_t cycles
Number of cycles to sleep
TBD, cycles used for measurement

uint32_t unused
Unused

uint32_t opcode
Opcode (OPCODE_DELAY)
Opcode (OPCODE_ST)
Opcode (OPCODE_LD)
Opcode (OPCODE_HALT)
Opcode (OPCODE_BRANCH)
Opcode (OPCODE_ALU)
Opcode (OPCODE_WR_REG)
Opcode (OPCODE_RD_REG)
Opcode (OPCODE_ADC)
Opcode (OPCODE_TSENS)
Opcode (OPCODE_I2C)
Opcode (OPCODE_END)
Opcode (OPCODE_MACRO)

struct ulp_insn::[anonymous] delay
Format of DELAY instruction
uint32_t dreg
  Register which contains data to store
  Register where the data should be loaded to
  Register which contains target PC, expressed in words (used if .reg == 1)
  Destination register
  Register where to store ADC result
  Register where to store temperature measurement result
  Destination register (for SUB_OPCODE_MACRO_LABELPC) >

uint32_t sreg
  Register which contains address in RTC memory (expressed in words)
  Register with operand A

uint32_t unused1
  Unused

uint32_t offset
  Offset to add to sreg
  Absolute value of target PC offset w.r.t. current PC, expressed in words

uint32_t unused2
  Unused

uint32_t sub_opcode
  Sub opcode (SUB_OPCODE_ST)
  Sub opcode (SUB_OPCODE_BX)
  Sub opcode (SUB_OPCODE_B)
  Sub opcode (SUB_OPCODE_BS)
  Sub opcode (SUB_OPCODE_ALU_REG)
  Sub opcode (SUB_OPCODE_ALU_CNT)
  Sub opcode (SUB_OPCODE_ALU_IMM)
  Sub opcode (SUB_OPCODE_WAKEUP)
  Sub opcode (SUB_OPCODE_SLEEP)
  SUB_OPCODE_MACRO_LABEL or SUB_OPCODE_MACRO_LABELPC
  SUB_OPCODE_MACRO_LABELPC or

struct ulp_insn::[anonymous] st
  Format of ST instruction

struct ulp_insn::[anonymous] ld
  Format of LD instruction

struct ulp_insn::[anonymous] halt
  Format of HALT instruction
uint32_t addr
Target PC, expressed in words (used if .reg == 0)
Address within either RTC_CNTL, RTC_IO, or SARADC

uint32_t reg
Target PC in register (1) or immediate (0)

uint32_t type
Jump condition (BX_JUMP_TYPE_xxx)

struct ulp_insn::{anonymous} bx
Format of BRANCH instruction (absolute address)

uint32_t imm
Immediate value to compare against
Immediate value of operand
Immediate value of operand B

uint32_t cmp
Comparison to perform: B_CMP_L or B_CMP_GE
Comparison to perform: JUMPS_LT, JUMPS_GE or JUMPS_LE

uint32_t sign
Sign of target PC offset: 0: positive, 1: negative

struct ulp_insn::{anonymous} b
Format of BRANCH instruction (relative address, conditional on R0)

struct ulp_insn::{anonymous} bs
Format of BRANCH instruction (relative address, conditional on the stage counter)

uint32_t treg
Register with operand B

uint32_t sel
Operation to perform, one of ALU_SEL_xxx
Operation to perform, one of ALU_SEL_Sxxx

struct ulp_insn::{anonymous} alu_reg
Format of ALU instruction (both sources are registers)

struct ulp_insn::{anonymous} alu_reg_s
Format of ALU instruction (stage counter and an immediate)

struct ulp_insn::{anonymous} alu_imm
Format of ALU instruction (one source is an immediate)
**Chapter 2. API 参考**

```c
uint32_t periph_sel
    Select peripheral: RTC_CNTL (0), RTC_IO (1), SARADC (2)

uint32_t data
    8 bits of data to write
    8 bits of data for write operation

uint32_t low
    Low bit

uint32_t high
    High bit

struct ulp_insn::[anonymous] wr_reg
    Format of WR_REG instruction

struct ulp_insn::[anonymous] rd_reg
    Format of RD_REG instruction

uint32_t mux
    Select SARADC pad (mux + 1)

uint32_t sar_sel
    Select SARADC0 (0) or SARADC1 (1)

struct ulp_insn::[anonymous] adc
    Format of ADC instruction

uint32_t wait_delay
    Cycles to wait after measurement is done

uint32_t reserved
    Reserved, set to 0

struct ulp_insn::[anonymous] tsens
    Format of TSENS instruction

uint32_t i2c_addr
    I2C slave address

uint32_t low_bits
    low bit of range for write operation (lower bits are masked)

uint32_t high_bits
    high bit of range for write operation (higher bits are masked)

uint32_t i2c_sel
    index of slave address register [7:0]
```
uint32_t rw
    Write (1) or read (0)

struct ulp_insn::[anonymous] i2c
    Format of I2C instruction

uint32_t wakeup
    Set to 1 to wake up chip

struct ulp_insn::[anonymous] end
    Format of END instruction with wakeup

uint32_t cycle_sel
    Select which one of SARADC_ULP_CP_SLEEP_CYCx_REG to get the sleep duration from

struct ulp_insn::[anonymous] sleep
    Format of END instruction with sleep

uint32_t label
    Label number

struct ulp_insn::[anonymous] macro
    Format of tokens used by MACROs

uint32_t instruction
    Encoded instruction for ULP coprocessor

**Macros**

R0
    general purpose register 0

R1
    general purpose register 1

R2
    general purpose register 2

R3
    general purpose register 3

OPCODE_WR_REG
    Instruction: write peripheral register (RTC_CNTL/RTC_IO/SARADC)

OPCODE_RD_REG
    Instruction: read peripheral register (RTC_CNTL/RTC_IO/SARADC)

RD_REG_PERIPH_RTC_CNTL
    Identifier of RTC_CNTL peripheral for RD_REG and WR_REG instructions
Chapter 2. API

**RD_REG_PERIPH_RTC_IO**
Identifier of RTC_IO peripheral for RD_REG and WR_REG instructions

**RD_REG_PERIPH_SENS**
Identifier of SARADC peripheral for RD_REG and WR_REG instructions

**RD_REG_PERIPH_RTC_I2C**
Identifier of RTC_I2C peripheral for RD_REG and WR_REG instructions

**OPCODE_I2C**
Instruction: read/write I2C

**SUB_OPCODE_I2C_RD**
I2C read

**SUB_OPCODE_I2C_WR**
I2C write

**OPCODE_DELAY**
Instruction: delay (nop) for a given number of cycles

**OPCODE_ADC**
Instruction: SAR ADC measurement

**OPCODE_ST**
Instruction: store indirect to RTC memory

**SUB_OPCODE_ST**
Store 32 bits, 16 MSBs contain PC, 16 LSBs contain value from source register

**OPCODE_ALU**
Arithmetic instructions

**SUB_OPCODE_ALU_REG**
Arithmetic instruction, both source values are in register

**SUB_OPCODE_ALU_IMM**
Arithmetic instruction, one source value is an immediate

**SUB_OPCODE_ALU_CNT**
Arithmetic instruction, stage counter and an immediate

**ALU_SEL_ADD**
Addition

**ALU_SEL_SUB**
Subtraction
Chapter 2. API Reference

**ALU_SEL_AND**
Logical AND

**ALU_SEL_OR**
Logical OR

**ALU_SEL_MOV**
Copy value (immediate to destination register or source register to destination register)

**ALU_SEL_LSH**
Shift left by given number of bits

**ALU_SEL_RSH**
Shift right by given number of bits

**ALU_SEL_SINC**
Increment the stage counter

**ALU_SEL_SDEC**
Decrement the stage counter

**ALU_SEL_SRST**
Reset the stage counter

**OPCODE_BRANCH**
Branch instructions

**SUB_OPCODE_BX**
Branch to absolute PC (immediate or in register)

**SUB_OPCODE_BR**
Branch to relative PC, conditional on R0

**SUB_OPCODE_BS**
Branch to relative PC, conditional on the stage counter

**BX_JUMP_TYPE_DIRECT**
Unconditional jump

**BX_JUMP_TYPE_ZERO**
Branch if last ALU result is zero

**BX_JUMP_TYPE_OVF**
Branch if last ALU operation caused and overflow

**SUB_OPCODE_B**
Branch to a relative offset
B_CMP_L
Branch if R0 is less than an immediate

B_CMP_GE
Branch if R0 is greater than or equal to an immediate

JUMPS_LT
Branch if the stage counter <

JUMPS_GE
Branch if the stage counter >=

JUMPS_LE
Branch if the stage counter <=

OPCODE_END
Stop executing the program

SUB_OPCODE_END
Stop executing the program and optionally wake up the chip

SUB_OPCODE_SLEEP
Stop executing the program and run it again after selected interval

OPCODE_TSENS
Instruction: temperature sensor measurement. Poor accuracy, not recommended for most use-cases

OPCODE_HALT
Halt the coprocessor

OPCODE_LD
Indirect load lower 16 bits from RTC memory

OPCODE_MACRO
Not a real opcode. Used to identify labels and branches in the program

SUB_OPCODE_MACRO_LABEL
Label macro

SUB_OPCODE_MACRO_BRANCH
Branch macro

SUB_OPCODE_MACRO_LABELPC
Label pointer macro

I_DELAY (cycles_)
Delay (nop) for a given number of cycles
I_HALT()
Halt the coprocessor.
This instruction halts the coprocessor, but keeps ULP timer active. As such, ULP program will be restarted again by timer. To stop the program and prevent the timer from restarting the program, use I_END(0) instruction.

I_WR_REG(reg, low_bit, high_bit, val)
Write literal value to a peripheral register
reg[high_bit : low_bit] = val This instruction can access RTC_CNTL_, RTC_IO_, SENS_, and RTC_I2C peripheral registers.

I_RD_REG(reg, low_bit, high_bit)
Read from peripheral register into R0
R0 = reg[high_bit : low_bit] This instruction can access RTC_CNTL_, RTC_IO_, SENS_, and RTC_I2C peripheral registers.

I_WR_REG_BIT(reg, shift, val)
Set or clear a bit in the peripheral register.
Sets bit (1 « shift) of register reg to value val. This instruction can access RTC_CNTL_, RTC_IO_, SENS_, and RTC_I2C peripheral registers.

I_WAKE()
Wake the SoC from deep sleep.
This instruction initiates wake up from deep sleep. Use esp_deep_sleep_enable_ulp_wakeup to enable deep sleep wakeup triggered by the ULP before going into deep sleep. Note that ULP program will still keep running until the I_HALT instruction, and it will still be restarted by timer at regular intervals, even when the SoC is woken up.
To stop the ULP program, use I_HALT instruction.
To disable the timer which start ULP program, use I_END() instruction. I_END instruction clears the RTC_CNTL_ULP_CP_SLP_TIMER_EN_S bit of RTC_CNTL_STATE0_REG register, which controls the ULP timer.

I_END()
Stop ULP program timer.
This is a convenience macro which disables the ULP program timer. Once this instruction is used, ULP program will not be restarted anymore until ulp_run function is called.
ULP program will continue running after this instruction. To stop the currently running program, use I_HALT().

I_SLEEP_CYCLE_SEL(timer_idx)
Select the time interval used to run ULP program.
This instructions selects which of the SENS_SLEEP_CYCLES_Sx registers’ value is used by the ULP program timer. When the ULP program stops at I_HALT instruction, ULP program timer start counting. When the counter reaches the value of the selected SENS_SLEEP_CYCLES_Sx register, ULP program start running again from the start address (passed to the ulp_run function). There are 5 SENS_SLEEP_CYCLES_Sx registers, so 0 <= timer_idx < 5.
By default, SENS_SLEEP_CYCLES_S0 register is used by the ULP program timer.

I_TSENS(reg_dest, delay)
Perform temperature sensor measurement and store it into reg_dest.
Delay can be set between 1 and ((1 « 14) - 1). Higher values give higher measurement resolution.
I_ADC (reg_dest, adc_idx, pad_idx)
Perform ADC measurement and store result in reg_dest.
adc_idx selects ADC (0 or 1). pad_idx selects ADC pad (0 - 7).

I_ST (reg_val, reg_addr, offset_)
Store value from register reg_val into RTC memory.
The value is written to an offset calculated by adding value of reg_addr register and offset_field (this offset is expressed in 32-bit words). 32 bits written to RTC memory are built as follows:
• bits [31:21] hold the PC of current instruction, expressed in 32-bit words
• bits [20:18] = 3’b0
• bits [17:16] reg_addr (0..3)
• bits [15:0] are assigned the contents of reg_val

RTC_SLOW_MEM[addr + offset_] = { insn_PC[10:0], 3’b0, reg_addr, reg_val[15:0] }

I_LD (reg_dest, reg_addr, offset_)
Load value from RTC memory into reg_dest register.
Loads 16 LSBS from RTC memory word given by the sum of value in reg_addr and value of offset_.

I_BL (pc_offset, imm_value)
Branch relative if R0 less than immediate value.
pc_offset is expressed in words, and can be from -127 to 127 imm_value is a 16-bit value to compare R0 against

I_BGE (pc_offset, imm_value)
Branch relative if R0 greater or equal than immediate value.
pc_offset is expressed in words, and can be from -127 to 127 imm_value is a 16-bit value to compare R0 against

I_BXR (reg_pc)
Unconditional branch to absolute PC, address in register.
reg_pc is the register which contains address to jump to. Address is expressed in 32-bit words.

I_BXI (imm_pc)
Unconditional branch to absolute PC, immediate address.
Address imm_pc is expressed in 32-bit words.

I_BXZR (reg_pc)
Branch to absolute PC if ALU result is zero, address in register.
reg_pc is the register which contains address to jump to. Address is expressed in 32-bit words.

I_BXZI (imm_pc)
Branch to absolute PC if ALU result is zero, immediate address.
Address imm_pc is expressed in 32-bit words.

I_BXFR (reg_pc)
Branch to absolute PC if ALU overflow, address in register
reg_pc is the register which contains address to jump to. Address is expressed in 32-bit words.

I_BXFI (imm_pc)
Branch to absolute PC if ALU overflow, immediate address
Address imm_pc is expressed in 32-bit words.

I_ADDR (reg_dest, reg_src1, reg_src2)
Addition: dest = src1 + src2
I_SUBR (reg_dest, reg_src1, reg_src2)
   Subtraction: dest = src1 - src2
I_ANDR (reg_dest, reg_src1, reg_src2)
   Logical AND: dest = src1 & src2
I_ORR (reg_dest, reg_src1, reg_src2)
   Logical OR: dest = src1 | src2
I_MOVR (reg_dest, reg_src)
   Copy: dest = src
I_LSHR (reg_dest, reg_src, reg_shift)
   Logical shift left: dest = src << shift
I_RSHR (reg_dest, reg_src, reg_shift)
   Logical shift right: dest = src >> shift
I_ADDI (reg_dest, reg_src, imm_)
   Add register and an immediate value: dest = src1 + imm
I_SUBI (reg_dest, reg_src, imm_)
   Subtract register and an immediate value: dest = src - imm
I_ANDI (reg_dest, reg_src, imm_)
   Logical AND register and an immediate value: dest = src & imm
I_ORI (reg_dest, reg_src, imm_)
   Logical OR register and an immediate value: dest = src | imm
I_MOVI (reg_dest, imm_)
   Copy an immediate value into register: dest = imm
I_LSHI (reg_dest, reg_src, imm_)
   Logical shift left register value by an immediate: dest = src << imm
I_RSHI (reg_dest, reg_src, imm_)
   Logical shift right register value by an immediate: dest = val >> imm

M_LABEL (label_num)
   Define a label with number label_num.
   This is a macro which doesn’t generate a real instruction. The token generated by this macro is removed by ulp_process_macros_and_load function. Label defined using this macro can be used in branch macros defined below.

M_BRANCH (label_num)
   Token macro used by M_B and M_BX macros. Not to be used directly.

M_LABELPC (label_num)
   Token macro used by M_MOVL macro. Not to be used directly.

M_MOVL (reg_dest, label_num)
   Macro: Move the program counter at the given label into the register. This address can then be used with I_BXR, I_BXZR, I_BXFR, etc.
   This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.
Chapter 2. API

M_BL (label_num, imm_value)
Macro: branch to label label_num if R0 is less than immediate value.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

M_BGE (label_num, imm_value)
Macro: branch to label label_num if R0 is greater or equal than immediate value.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

M_BX (label_num)
Macro: unconditional branch to label.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

M_BXZ (label_num)
Macro: branch to label if ALU result is zero.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

M_BXF (label_num)
Macro: branch to label if ALU overflow.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

I_STAGE_INC (imm_)
Increment the stage counter by immediate value.

I_STAGE_DEC (imm_)
Decrement the stage counter by immediate value.

I_STAGE_RST ()
Reset the stage counter.

M_BSLT (label_num, imm_value)
Macro: branch to label if the stage counter is less than immediate value.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

M_BSGE (label_num, imm_value)
Macro: branch to label if the stage counter is greater than or equal to immediate value.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

M_BSLE (label_num, imm_value)
Macro: branch to label if the stage counter is less than or equal to immediate value.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.
**M_BSEQ** (label_num, imm_value)

Macro: branch to label if the stage counter is equal to immediate value. Implemented using two JUMPS instructions: JUMPS next, imm_value, LT JUMPS label_num, imm_value, LE

This macro generates three ulp_insn_t values separated by commas, and should be used when defining contents of ulp_insn_t arrays. Second value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

**M_BSGT** (label_num, imm_value)

Macro: branch to label if the stage counter is greater than immediate value. Implemented using two instructions: JUMPS next, imm_value, LE JUMPS label_num, imm_value, GE

This macro generates three ulp_insn_t values separated by commas, and should be used when defining contents of ulp_insn_t arrays. Second value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

**I_JUMPS** (pc_offset, imm_value, comp_type)

Branch relative if (stage counter [comp_type] imm_value) evaluates to true.

pc_offset is expressed in words, and can be from -127 to 127 imm_value is an 8-bit value to compare the stage counter against comp_type is the type of comparison to perform: JUMPS_LT (<), JUMPS_GE (>=) or JUMPS_LE (<=)

**I_I2C_RW** (sub_addr, val, low_bit, high_bit, slave_sel, rw_bit)

Perform an I2C transaction with a slave device. I_I2C_READ and I_I2C_WRITE are provided for convenience, instead of using this directly.

Slave address (in 7-bit format) has to be set in advance into SENS_I2C_SLAVE_ADDRx register field, where x == slave_sel. For read operations, 8 bits of read result is stored into R0 register. For write operations, val will be written to sub_addr at [high_bit:low_bit]. Bits outside of this range are masked.

**I_I2C_READ** (slave_sel, sub_addr)

Read a byte from the sub address of an I2C slave, and store the result in R0.

Slave address (in 7-bit format) has to be set in advance into SENS_I2C_SLAVE_ADDRx register field, where x == slave_sel.

**I_I2C_WRITE** (slave_sel, sub_addr, val)

Write a byte to the sub address of an I2C slave.

Slave address (in 7-bit format) has to be set in advance into SENS_I2C_SLAVE_ADDRx register field, where x == slave_sel.

---

**编译 ULP 代码**

若需要将 ULP FSM 代码编译为某组件的一部分，则必须执行以下步骤：

1. 用汇编语言编写的 ULP FSM 代码必须导入到一个或多个 .S 扩展文件中，且这些文件必须放在组件目录中一个独立的目录中，例如 ulp/。
2. 注册后从组件 CMakeLists.txt 中调用 ulp_embed_binary 示例如下:

```cpp
... idf_component_register()

set(ulp_app_name ulp_${COMPONENT_NAME})
set(ulp_s_sources ulp/ulp_assembly_source_file.S)
set(ulp_exp_dep_srcs "ulp_c_source_file.c")

ulp_embed_binary(${ulp_app_name} "${ulp_s_sources}" "${ulp_exp_dep_srcs}")
```

ulp_embed_binary 的第一个参数为 ULP 二进制文件名。指定的此名称也用于生成的其他文件，如: ELF 文件, .map 文件, 头文件和链接器导出文件。第二个参数指定 ULP FSM 程序集源文件。最后，第
三个参数指定组件源文件列表，其中包括被生成的头文件。此列表用以建立正确的依赖项，并确保在编译这些文件之前先创建生成的头文件。有关 ULP FSM 应用程序生成的头文件等其他概念，请参考下文。

3. 使用常规方法（例如 idl.py app）编译应用程序。
   在内部，构建系统将按照以下步骤编译 ULP FSM 程序:
   1. 通过 C 预处理器运行每个程序集文件 (foo.S)。此步骤在组件编译目录中生成预处理的程序集文件 (foo.ulp.S)，同时生成依赖文件 (foo.ulp.d)。
   2. 通过汇编器运行预处理过的汇编源码。此步骤会生成目标文件 (foo.ulp.o) 和清单 (foo.ulp.lst)。清单文件仅用于调试，不用于编译过程的后续步骤。
   3. 通过 C 预处理器运行链接器脚本模板。模板位于 components/ulp/ld 目录中。
   4. 将目标文件链接到 ELF 输出文件 (ulp_app_name.elf)。此步骤生成的.map 文件 (ulp_app_name.map) 默认用于调试。
   5. 将 ELF 文件中的内容转储为二进制文件 (ulp_app_name.bin)，以便嵌入到应用程序中。
   6. 使用 esp32ulp-elf-nm 在 ELF 文件中生成全局符号列表 (ulp_app_name.sym)。
   7. 创建 LD 导出脚本和头文件 (ulp_app_name.ld 和 ulp_app_name.h)，包含来自 ulp_app_name.sym 的符号。此步骤可借助 esp32ulp_mapgen.py 工具来完成。
   8. 将生成的二进制文件添加到要嵌入应用程序的二进制文件列表中。

访问 ULP FSM 程序变量

在 ULP FSM 程序中定义的全局符号也可以在主程序中使用。
例如，ULP FSM 程序可以定义 measurement_count 变量，此变量可以定义程序从深度睡眠中唤醒芯片之前需要进行的 ADC 测量的次数。

```
.globa l measurement_count

// later, use measurement_count
move r3, measurement_count
ld r3, r3, 0
```

主程序需要在启动 ULP 程序之前初始化 measurement_count 变量，构建系统通过生成定义 ULP 编程中的全局符号的 $(ULP_APP_NAME).h 和 $(ULP_APP_NAME).ld 文件实现上述操作。这些文件包含了在 ULP 程序中定义的所有全局符号，文件以 ulp_ 开头。

头文件包含对此类符号的声明:

```
extern uint32_t ulp_measurement_count;
```

注意，所有符号（包括变量、数组、函数）均被声明为 uint32_t，对于函数和数组，先获取符号地址，然后转换为适当的类型。

生成的链接器脚本文件定义了 RTC_SLOW_MEM 中的符号位置:

```
PROVIDE ( ulp_measurement_count = 0x50000060 );
```

如果要从主程序访问 ULP 程序变量，应先使用 include 语句包含生成的头文件，这样，就可以像访问常规变量一样访问 ulp 程序变量。操作如下:

```
#include "ulp_app_name.h"

// later
void init_ulp_vars() {
    ulp_measurement_count = 64;
}
```

注意，ULP FSM 程序在 RTC 内存中只能使用 32 位字的低 16 位，因为寄存器是 16 位的，并且不具备从字的高位加载的指令。同样，ULP 储存指令将寄存器值写入 32 位字的低 16 位中。高 16 位写入的值取决于存储指令的地址，因此在读取 ULP 协处理器写的变量时，主应用程序需要屏蔽高 16 位，例如:
启动 ULP FSM 程序

要运行 ULP FSM 程序，主应用程序需要调用 `ulp_load_binary()` 函数将 ULP 程序加载到 RTC 内存中，然后调用 `ulp_run()` 函数，启动 ULP 程序。

注意，在 menuconfig 中必须启用 Enable Ultra Low Power (ULP) Co-processor 选项，以便正常运行 ULP，并且必须设置 ULP Co-processor type 选项，以便选择要使用的 ULP 类型。RTC slow memory reserved for coprocessor 选项设置的值必须足够存储 ULP 代码和数据。如果应用程序组件包含多个 ULP 程序，则 RTC 内存必须足以容纳最大的程序。

每个 ULP 程序均以 BLOB 的形式嵌入到 ESP-IDF 应用程序中。应用程序可以引用此 BLOB，并以下面的方式加载此 BLOB（假设 ULP_APP_NAME 已被定义为 ulp_app_name）：

```c
extern const uint8_t bin_start[] asm("_binary_ulp_app_name_bin_start");
extern const uint8_t bin_end[] asm("_binary_ulp_app_name_bin_end");
void start_ulp_program() {
  ESP_ERROR_CHECK( ulp_load_binary(
      0 /* load address, set to 0 when using default linker scripts
      bin_start,
      (bin_end - bin_start) / sizeof(uint32_t)) );
}
```

一旦上述程序加载到 RTC 内存后，应用程序即可启动此程序，并将入口点的地址传递给 `ulp_run` 函数：

```c
ESP_ERROR_CHECK( ulp_run(&ulp_entry - RTC_SLOW_MEM ) );
```

上述生成的头文件 `$<ULP_APP_NAME>.h` 声明了入口点符号。在 ULP 应用程序的汇编源代码中，此符号必须标记为 `.global`:

```asm
.global entry
entry: // code starts here
```

ESP32 ULP 程序流程

ESP32 ULP 协处理器由定时器启动，而调用 `ulp_run()` 则可启动此定时器。定时器为 RTC_SLOW_CLK 的 Tick 事件计数（默认情况下，Tick 由内部 150 KHz RC 振荡器生成）。使用 SENS_ULP_CP_SLEEP_CYC0_REG 寄存器 (x = 0.4) 设置 Tick 数值。第一次启动 ULP 时，使用 SENS_ULP_CP_SLEEP_CYC0_REG 设置定时器 Tick 数值，之后，ULP 程序可以使用 sleep 指令来选择另一个 SENS_ULP_CP_SLEEP_CYC0_REG 寄存器。

此应用程序可以调用 `ulp_set_wakeup_period` 函数来设置 ULP 定时器周期值 (SENS_ULP_CP_SLEEP_CYC0_REG, x = 0.4)。

一旦定时器计数到 SENS_ULP_CP_SLEEP_CYC0_REG 寄存器设定的 Tick 数值，ULP 协处理器就会启动，并调用 `ulp_run()` 的入口点开始运行程序。

程序保持运行，直到遇到 halt 指令或非法指令。一旦程序停止，ULP 协处理器电源关闭，定时器再次启动。

如果想禁用定时器（有效防止 ULP 程序再次运行），可在 ULP 代码或主程序中清除 RTC_CNTL_STATE0_REG 寄存器中的 RTC_CNTL_ULP_CP_SLP_TIMER_EN 位。

应用示例

- 主处理器处于深睡眠状态时，ULP FSM 协处理器对 IO 脉冲进行计数：`system/ulp_fsm/ulp`。

Espressif Systems 2095 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
API 参考

Header File

- components/ulp/ulp_fsm/include/ulp_fsm_common.h

Functions

`esp_err_t ulp_process_macros_and_load (uint32_t load_addr, const ulp_insn_t *program, size_t *psize)`

Resolve all macro references in a program and load it into RTC memory.

参数
- `load_addr` – address where the program should be loaded, expressed in 32-bit words
- `program` – ulp_insn_t array with the program
- `psize` – size of the program, expressed in 32-bit words

返回
- ESP_OK on success
- ESP_ERR_NO_MEM if auxiliary temporary structure cannot be allocated
- one of ESP_ERR_ULP_xxx if program is not valid or cannot be loaded

`esp_err_t ulp_load_binary (uint32_t load_addr, const uint8_t *program_binary, size_t program_size)`

Load ULP program binary into RTC memory.

ULP program binary should have the following format (all values little-endian):

a. MAGIC, (value 0x00706c75, 4 bytes)
b. TEXT_OFFSET, offset of .text section from binary start (2 bytes)
c. TEXT_SIZE, size of .text section (2 bytes)
d. DATA_SIZE, size of .data section (2 bytes)
e. BSS_SIZE, size of .bss section (2 bytes)
f. (TEXT_OFFSET - 12) bytes of arbitrary data (will not be loaded into RTC memory)
g. .text section
h. .data section

Linker script in components/ulp/ld/esp32.ulp.ld produces ELF files which correspond to this format. This linker script produces binaries with load_addr == 0.

参数
- `load_addr` – address where the program should be loaded, expressed in 32-bit words
- `program_binary` – pointer to program binary
- `program_size` – size of the program binary

返回
- ESP_OK on success
- ESP_ERR_INVALID_ARG if load_addr is out of range
- ESP_ERR_INVALID_SIZE if program_size doesn’t match (TEXT_OFFSET + TEXT_SIZE + DATA_SIZE)
- ESP_ERR_NOT_SUPPORTED if the magic number is incorrect

`esp_err_t ulp_run (uint32_t entry_point)`

Run the program loaded into RTC memory.

参数 `entry_point` – entry point, expressed in 32-bit words

返回 ESP_OK on success

Macros

`ESP_ERR_ULP_BASE`

Offset for ULP-related error codes

- 主处理器处于 Deep-sleep 状态时，ULP FSM 协处理器轮询 ADC：system/ulp_fsm/ulp_adc。
Chapter 2. API

ESP_ERR_ULP_SIZE_TOO_BIG
Program doesn’t fit into RTC memory reserved for the ULP

ESP_ERR_ULP_INVALID_LOAD_ADDR
Load address is outside of RTC memory reserved for the ULP

ESP_ERR_ULP_DUPLICATE_LABEL
More than one label with the same number was defined

ESP_ERR_ULP_UNDEFINED_LABEL
Branch instructions references an undefined label

ESP_ERR_ULP_BRANCH_OUT_OF_RANGE
Branch target is out of range of B instruction (try replacing with BX)

Type Definitions
typedef union ulp_insn ulp_insn_t

Header File
- components/ulp/ulp_common/include/ulp_common.h

Functions
esp_err_t ulp_set_wakeup_period (size_t period_index, uint32_t period_us)
Set one of ULP wakeup period values.

ULP coprocessor starts running the program when the wake up timer counts up to a given value (called period). There are 5 period values which can be programmed into SENS_ULP_CP_SLEEP_CYCx_REG registers, x = 0..4 for ESP32, and one period value which can be programmed into RTC_CNTL_ULP_CP_TIMER_1_REG register for ESP32-S2/S3. By default, for ESP32, wake up timer will use the period set into SENS_ULP_CP_SLEEP_CYC0_REG, i.e. period number 0. ULP program code can use SLEEP instruction to select which of the SENS_ULP_CP_SLEEP_CYCx_REG should be used for subsequent wakeups.

However, please note that SLEEP instruction issued (from ULP program) while the system is in deep sleep mode does not have effect, and sleep cycle count 0 is used.

For ESP32-S2/S3 the SLEEP instruction not exist. Instead a WAKE instruction will be used.

Remarks: The ULP FSM requires two clock cycles to wake up before being able to run the program. Then additional 16 cycles are reserved after wake up waiting until the SM clock is stable. The FSM also requires two more clock cycles to go to sleep after the program execution is halted. The minimum wakeup period that may be set up for the ULP is equal to the total number of cycles spent on the above internal tasks. For a default configuration of the ULP running at 150kHz it makes about 133us.

Parameters
- period_index – wake up period setting number (0 - 4)
- period_us – wake up period, us

Returns
- ESP_OK on success
- ESP_ERR_INVALID_ARG if period_index is out of range
void ulp_timer_stop (void)
    Stop the ULP timer.

    备注： This will stop the ULP from waking up if halted, but will not abort any program currently executing on the ULP.

void ulp_timer_resume (void)
    Resume the ULP timer.

    备注： This will resume an already configured timer, but does no other configuration

Header File

- components/ulp/ulp_common/include/esp32/ulp_common_defs.h

Macros

RTC_SLOW_MEM
    RTC slow memory, 8k size

2.10.30 Watchdogs

Overview

The ESP-IDF has support for multiple types of watchdogs, with the two main ones being: The Interrupt Watchdog Timer and the Task Watchdog Timer (TWDT). The Interrupt Watchdog Timer and the TWDT can both be enabled using Project Configuration Menu, however the TWDT can also be enabled during runtime. The Interrupt Watchdog is responsible for detecting instances where FreeRTOS task switching is blocked for a prolonged period of time. The TWDT is responsible for detecting instances of tasks running without yielding for a prolonged period.

ESP-IDF has support for the following types of watchdog timers:

- Interrupt Watchdog Timer (IWDT)
- Task Watchdog Timer (TWDT)

The various watchdog timers can be enabled using the Project Configuration Menu. However, the TWDT can also be enabled during runtime.

Interrupt Watchdog Timer (IWDT)

{IDF_IWDT_TIMER_GROUP:default=“ Timer Group 1” , esp32c2=“ Timer Group 0” }

The purpose of the IWDT is to ensure that interrupt service routines (ISRs) are not blocked from running for a prolonged period of time (i.e., the IWDT timeout period). Blocking ISRs from running in a timely manner is undesirable as it can increase ISR latency, and also prevents task switching (as task switching is executed form an ISR). The things that can block ISRs from running include:

- Disabling interrupts
- Critical Sections (also disables interrupts)
- Other same/higher priority ISRs (will block same/lower priority ISRs from running it completes execution)
The IWDT utilizes the watchdog timer in {IDF_IWDT_TIMER_GROUP} as its underlying hardware timer and leverages the FreeRTOS tick interrupt on each CPU to feed the watchdog timer. If the tick interrupt on a particular CPU is not run at within the IWDT timeout period, it is indicative that something is blocking ISRs from being run on that CPU (see the list of reasons above).

When the IWDT times out, the default action is to invoke the panic handler and display the panic reason as Interrupt wdt timeout on CPU0 or Interrupt wdt timeout on CPU1 (as applicable). Depending on the panic handler’s configured behavior (see CONFIG_ESP_SYSTEM_PANIC), users can then debug the source of the IWDT timeout (via the backtrace, OpenOCD, gdbstub etc) or simply reset the chip (which may be preferred in a production environment).

If for whatever reason the panic handler is unable to run after an IWDT timeout, the IWDT has a secondary timeout that will hard-reset the chip (i.e., a system reset).

**Configuration**

- The IWDT is enabled by default via the `CONFIG_ESP_INT_WDT` option.
- The IWDT’s timeout is configured by setting the `CONFIG_ESP_INT_WDT_TIMEOUT_MS` option.
  - Note that the default timeout is higher if PSRAM support is enabled, as a critical section or interrupt routine that accesses a large amount of PSRAM will take longer to complete in some circumstances.
  - The timeout should always at least twice longer than the period between FreeRTOS ticks (see `CONFIG_FREERTOS_HZ`).

**Tuning** If you find the IWDT timeout is triggered because an interrupt or critical section is running longer than the timeout period, consider rewriting the code:

- Critical sections should be made as short as possible. Any non-critical code/computation should be placed outside the critical section.
- Interrupt handlers should also perform the minimum possible amount of computation. Users can consider deferring any computation to a task by having the ISR push data to a task using queues.

Neither critical sections or interrupt handlers should ever block waiting for another event to occur. If changing the code to reduce the processing time is not possible or desirable, it’s possible to increase the `CONFIG_ESP_INT_WDT_TIMEOUT_MS` setting instead.

**Task Watchdog Timer (TWDT)**

The Task Watchdog Timer (TWDT) is used to monitor particular tasks, ensuring that they are able to execute within a given timeout period. The TWDT primarily watches the Idle Tasks of each CPU, however any task can subscribe to be watched by the TWDT. By watching the Idle Tasks of each CPU, the TWDT can detect instances of tasks running for a prolonged period of time without yielding. This can be an indicator of poorly written code that spinloops on a peripheral, or a task that is stuck in an infinite loop.

The TWDT is built around the Hardware Watchdog Timer in Timer Group 0. When a timeout occurs, an interrupt is triggered. Users can define the function `esp_task_wdt_isr_user_handler` in the user code, in order to receive the timeout event and extend the default behavior.

**Usage** The following functions can be used to watch tasks using the TWDT:

- `esp_task_wdt_init()` to initialize the TWDT and subscribe the idle tasks.
- `esp_task_wdt_add()` subscribes other tasks to the TWDT.
- Once subscribed, `esp_task_wdt_reset()` should be called from the task to feed the TWDT.
- `esp_task_wdt_delete()` unsubsribes a previously subscribed task
- `esp_task_wdt_deinit()` unsubscribes the idle tasks and deinitializes the TWDT

In the case where applications need to watch at a more granular level (i.e., ensure that a particular functions/stub/code-path is called), the TWDT allows subscription of “users”.

- `esp_task_wdt_add_user()` to subscribe an arbitrary user of the TWDT. This function will return a user handle to the added user.
Chapter 2. API

- `esp_task_wdt_reset_user()` must be called using the user handle in order to prevent a TWDT timeout.
- `esp_task_wdt_delete_user()` unsubscribes an arbitrary user of the TWDT.

**Configuration** The default timeout period for the TWDT is set using config item `CONFIG_ESP_TASK_WDT_TIMEOUT_S`. This should be set to at least as long as you expect any single task will need to monopolize the CPU (for example, if you expect the app will do a long intensive calculation and should not yield to other tasks). It is also possible to change this timeout at runtime by calling `esp_task_wdt_init()`.

备注: Erasing large flash areas can be time consuming and can cause a task to run continuously, thus triggering a TWDT timeout. The following two methods can be used to avoid this:

- Increase `CONFIG_ESP_TASK_WDT_TIMEOUT_S` in menuconfig for a larger watchdog timeout period.
- You can also call `esp_task_wdt_init()` to increase the watchdog timeout period before erasing a large flash area.

For more information, you can refer to **SPI Flash**.

The following config options control TWDT configuration. They are all enabled by default:

- `CONFIG_ESP_TASK_WDT_EN` - enables TWDT feature. If this option is disabled, TWDT cannot be used, even if initialized at runtime.
- `CONFIG_ESP_TASK_WDT_INIT` - the TWDT is initialized automatically during startup. If this option is disabled, it is still possible to initialize the Task WDT at runtime by calling `esp_task_wdt_init()`.
- `CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0` - CPU0 Idle task is subscribed to the TWDT during startup. If this option is disabled, it is still possible to subscribe the idle task by calling `esp_task_wdt_init()` again.
- `CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1` - CPU1 Idle task is subscribed to the TWDT during startup.

**JTAG & Watchdogs**

While debugging using OpenOCD, the CPUs will be halted every time a breakpoint is reached. However if the watchdog timers continue to run when a breakpoint is encountered, they will eventually trigger a reset making it very difficult to debug code. Therefore OpenOCD will disable the hardware timers of both the interrupt and task watchdogs at every breakpoint. Moreover, OpenOCD will not reenable them upon leaving the breakpoint. This means that interrupt watchdog and task watchdog functionality will essentially be disabled. No warnings or panics from either watchdogs will be generated when the ESP32 is connected to OpenOCD via JTAG.

**API Reference**

**Task Watchdog** A full example using the Task Watchdog is available in esp-idf: `system/task_watchdog`

**Header File**

- `components/esp_system/include/esp_task_wdt.h`

**Functions**

```c
esp_err_t esp_task_wdt_init(const esp_task_wdt_config_t *config)
```

Initialize the Task Watchdog Timer (TWDT)

This function configures and initializes the TWDT. This function will subscribe the idle tasks if configured to do so. For other tasks, users can subscribe them using `esp_task_wdt_add()` or `esp_task_wdt_add_user()`. This function won’t start the timer if no task have been registered yet.
Chapter 2. API 参考

备注: esp_task_wdt_init() must only be called after the scheduler is started. Moreover, it must not be called by multiple tasks simultaneously.

参数 config [in] Configuration structure
返回
• ESP_OK: Initialization was successful
• ESP_ERR_INVALID_STATE: Already initialized
• Other: Failed to initialize TWDT

esp_err_t esp_task_wdt_reconfigure(const esp_task_wdt_config_t *config)
Reconfigure the Task Watchdog Timer (TWDT)
The function reconfigures the running TWDT. It must already be initialized when this function is called.

备注: esp_task_wdt_reconfigure() must not be called by multiple tasks simultaneously.

参数 config [in] Configuration structure
返回
• ESP_OK: Reconfiguring was successful
• ESP_ERR_INVALID_STATE: TWDT not initialized yet
• Other: Failed to initialize TWDT

esp_err_t esp_task_wdt_deinit(void)
Deinitialize the Task Watchdog Timer (TWDT)
This function will deinitialize the TWDT, and unsubscribe any idle tasks. Calling this function whilst other tasks are still subscribed to the TWDT, or when the TWDT is already deinitialized, will result in an error code being returned.

备注: esp_task_wdt_deinit() must not be called by multiple tasks simultaneously.

返回
• ESP_OK: TWDT successfully deinitialized
• Other: Failed to deinitialize TWDT

esp_err_t esp_task_wdt_add(TaskHandle_t task_handle)
Subscribe a task to the Task Watchdog Timer (TWDT)
This function subscribes a task to the TWDT. Each subscribed task must periodically call esp_task_wdt_reset() to prevent the TWDT from elapsing its timeout period. Failure to do so will result in a TWDT timeout.

参数 task_handle — Handle of the task. Input NULL to subscribe the current running task to the TWDT
返回
• ESP_OK: Successfully subscribed the task to the TWDT
• Other: Failed to subscribe task

esp_err_t esp_task_wdt_add_user(const char *user_name, esp_task_wdt_user_handle_t *user_handle_ret)
Subscribe a user to the Task Watchdog Timer (TWDT)
This function subscribes a user to the TWDT. A user of the TWDT is usually a function that needs to run periodically. Each subscribed user must periodically call esp_task_wdt_reset_user() to prevent the TWDT from elapsing its timeout period. Failure to do so will result in a TWDT timeout.

参数
Chapter 2. API

返回
• ESP_OK: Successfully subscribed the user to the TWDT
• Other: Failed to subscribe user

esp_err_t esp_task_wdt_reset (void)
Reset the Task Watchdog Timer (TWDT) on behalf of the currently running task.

This function will reset the TWDT on behalf of the currently running task. Each subscribed task must periodically call this function to prevent the TWDT from timing out. If one or more subscribed tasks fail to reset the TWDT on their own behalf, a TWDT timeout will occur.

返回
• ESP_OK: Successfully reset the TWDT on behalf of the currently running task
• Other: Failed to reset

def esp_task_wdt_reset_user (esp_task_wdt_user_handle_t user_handle)
Reset the Task Watchdog Timer (TWDT) on behalf of a user.

This function will reset the TWDT on behalf of a user. Each subscribed user must periodically call this function to prevent the TWDT from timing out. If one or more subscribed users fail to reset the TWDT on their own behalf, a TWDT timeout will occur.

参数 user_handle – [in] User handle
• ESP_OK: Successfully reset the TWDT on behalf of the user
• Other: Failed to reset

esp_err_t esp_task_wdt_delete (TaskHandle_t task_handle)
Unsubscribe a task from the Task Watchdog Timer (TWDT)

This function will unsubscribe a task from the TWDT. After being unsubscribed, the task should no longer call esp_task_wdt_reset().

参数 task_handle – [in] Handle of the task. Input NULL to unsubscribe the current running task.

返回
• ESP_OK: Successfully unsubscribed the task from the TWDT
• Other: Failed to unsubscribe task

esp_err_t esp_task_wdt_delete_user (esp_task_wdt_user_handle_t user_handle)
Unsubscribe a user from the Task Watchdog Timer (TWDT)

This function will unsubscribe a user from the TWDT. After being unsubscribed, the user should no longer call esp_task_wdt_reset_user().

参数 user_handle – [in] User handle

返回
• ESP_OK: Successfully unsubscribed the user from the TWDT
• Other: Failed to unsubscribe user

esp_err_t esp_task_wdt_status (TaskHandle_t task_handle)
Query whether a task is subscribed to the Task Watchdog Timer (TWDT)

This function will query whether a task is currently subscribed to the TWDT, or whether the TWDT is initialized.

参数 task_handle – [in] Handle of the task. Input NULL to query the current running task.

返回
• ESP_OK: The task is currently subscribed to the TWDT
• ESP_ERR_NOT_FOUND: The task is not subscribed
• ESP_ERR_INVALID_STATE: TWDT was never initialized
Chapter 2. API

void esp_task_wdt_isr_user_handler (void)
User ISR callback placeholder.
This function is called by task_wdt_isr function (ISR for when TWDT times out). It can be defined in user
code to handle TWDT events.

备注: It has the same limitations as the interrupt function. Do not use ESP_LOGx functions inside.

Structures

struct esp_task_wdt_config_t
Task Watchdog Timer (TWDT) configuration structure.

Public Members

uint32_t timeout_ms
TWDT timeout duration in milliseconds

uint32_t idle_core_mask
Mask of the cores who’s idle task should be subscribed on initialization

bool trigger_panic
Trigger panic when timeout occurs

Type Definitions

typedef struct esp_task_wdt_user_handle_s *esp_task_wdt_user_handle_t
Task Watchdog Timer (TWDT) user handle.

此部分 API 代码示例存放在 ESP-IDF 示例项目的 system 目录下。
Chapter 3

H/W 硬件参考
Chapter 4

API 指南

4.1 应用层跟踪库

4.1.1 概述

ESP-IDF 中提供了应用层跟踪功能，用于分析应用程序的行为。这一功能在相应的库中实现，可以通过 menuconfig 开启。此功能允许用户在程序运行开销很小的前提下，通过 JTAG、UART 或 USB 接口在主机和 ESP32 之间传输任意数据。用户也可同时使用 JTAG 和 UART 接口。UART 接口主要用于连接 SEGGER SystemView 工具（参见 SystemView）。

开发人员可以使用这一功能库将应用程序的运行状态发送给主机，在运行时接收来自主机的命令或者其他类型的信息。该库的主要使用场景有：

1. 收集来自特定应用程序的数据，具体请参阅特定应用程序的跟踪。
2. 记录到主机的轻量级日志，具体请参阅记录日志到主机。
3. 系统行为分析，具体请参阅基于 SEGGER SystemView 的系统行为分析。
4. 获取源代码覆盖率，具体请参阅 Gcov（源代码覆盖）。

使用 JTAG 接口的跟踪组件工作示意图如下所示：

4.1.2 运行模式

该库支持两种运行模式：

后验模式：后验模式为默认模式，该模式不需和主机进行交互。在这种模式下，跟踪模块不会检查主机是否已经从 HW UP BUFFER 冲突区读走所有数据，而是直接使用新数据覆盖旧数据。如果用户仅对最新的跟踪数据感兴趣，例如需要分析程序在崩溃之前的行为，则推荐使用此模式。主机可以稍后根据用户的请求来读取数据，例如在使用 JTAG 接口的情况下，通过特殊的 OpenOCD 命令进行读取。

流模式：当主机连接到 ESP32 时，跟踪模块会进入此模式。在这种模式下，跟踪模块在新数据写入 HW UP BUFFER 之前会检查其是否有足够的空间。并在必要的时候等待主机读取数据并释放足够的内存。最大等待时间是由用户传递给相应 API 函数的超时时间参数决定的。因此当应用程序尝试使用有限的最大等待时间值来获取数据并使用缓冲区时，这些数据可能会被丢弃。尤其需要注意的是，如果在对实时要求严格的代码中（如中断处理函数、操作系统调度等）指定了无限的超时时间，将会导致系统故障。为了避免丢失此类关键数据，开发人员可以在 menuconfig 中开启 CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX 选项，以启用额外的数据缓冲区。此宏还指定了在上述条件下可以缓冲的数据大小，它有助于缓解由于 USB 总线拥塞等原因导致的向主机传输数据间歇性减缓的状况。但是，当跟踪数据流的平均比特率超出硬件接口的能力时，该选项无法发挥作用。

2107
4.1.3 配置选项与依赖项

使用此功能需要在主机端和目标端进行以下配置:

1. **主机端**: 应用程序跟踪通过 JTAG 来完成，因此需要在主机上安装并运行 OpenOCD。详细信息请参阅 JTAG 调试。

2. **目标端**: 在 menuconfig 中开启应用程序跟踪功能。前往 Component config > Application Level Tracing 菜单，选择跟踪数据的传输目标 (具体用于传输的硬件接口: JTAG 和/或 UART)，选择任一非 None 的目标都会启动开启 CONFIG_APPTRACE_ENABLE 选项。对于 UART 接口，用户必须定义波特率、TX 和 RX 管脚以及其他相关参数。

**备注**: 为了实现更高的数据速率并降低丢包率，建议优化 JTAG 的时钟频率，使其达到能够稳定运行的最大值。详细信息请参阅优化 JTAG 的速度。

以下为前述未提及的另外两个 menuconfig 选项:

1. **Threshold for flushing last trace data to host on panic (CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH)**。使用 JTAG 接口时，此选项是必选项。在该模式下，跟踪数据以 16 KB 数据块的形式暴露给主机。在后验模式中，一个块被填充后会被暴露给主机，同时之前的块不再可用。也就是说，跟踪数据以 16 KB 的粒度进行覆盖。发生 Panic 时，当前输入块的最新数据将会被暴露给主机，主机可以读取数据以进行后续分析。如果系统发生 Panic 时，仍有少量数据还来得及暴露给主机，那么之前收集的 16 KB 数据将丢失。主机只能获取少部分的最新跟踪数据，从而可能无法诊断问题。此 menuconfig 选项有助于避免此类情况，它控制发生 Panic 时刷新数据的阈值。例如，用户可能设置需要不少于 512 字节的跟踪数据，如果在发生 Panic 时待处理的数据少于 512 字节，则数据不会被刷新，也不会覆盖之前的 16 KB 数据。该选项仅在后验模式和使用 JTAG 工作时可发挥作用。

2. **Timeout for flushing last trace data to host on panic (CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO)**。该选项仅在流模式下才可发挥作用，它可用于控制跟踪模块在发生 Panic 时等待主机读取跟踪数据的最长时间。

3. **UART RX/TX ring buffer size (CONFIG_APPTRACE_UART_TX_BUFF_SIZE)**。缓冲区的大小取决于通过 UART 传输的数据量。
4. UART TX message size (ref:CONFIG_APPTRACE_UART_TX_MSG_size)。要传输的单条消息的最大尺寸。

4.1.4 如何使用此库

该库提供了用于在主机和 ESP32 之间传输任意数据的 API。在 menuconfig 中启用该库后，目标应用程序的跟踪模块会在系统启动时自动初始化。因此，用户需要做的就是调用相应的 API 来发送、接收或者刷新数据。

特定应用程序的跟踪

通常，用户需要决定在每个方向上传输数据的类型以及如何解析（处理）这些数据。要想在目标和主机之间传输数据，则需执行以下几个步骤：

1. 在目标端，用户需要实现将跟踪数据写入主机的算法。下面的代码片段展示了如何执行此操作。

```c
#include "esp_app_trace.h"
...
char buf[] = "Hello World!";
esp_err_t res = esp_apptrace_write(ESP_APTRACE_DEST_TRAX, buf, strlen(buf),
                                 ESP_APTRACE_TMO_INFINITE);
if (res != ESP_OK) {
    ESP_LOGE(TAG, "Failed to write data to host!");
    return res;
}
```

esp_apptrace_write() 函数使用 memcpy 把用户数据复制到内部缓存中。在某些情况下，使用 esp_apptrace_buffer_get() 和 esp_apptrace_buffer_put() 函数会更加理想，它们允许开发人员自行分配缓冲区并填充。下面的代码片段展示了如何执行此操作。

```c
#include "esp_app_trace.h"
...
int number = 10;
char *ptr = (char *)esp_apptrace_buffer_get(ESP_APTRACE_DEST_TRAX, 32, 100/
                                          "tmo in us");
if (ptr == NULL) {
    ESP_LOGE(TAG, "Failed to get buffer!");
    return ESP_FAIL;
}
sprintf(ptr, "Here is the number %d", number);
esp_err_t res = esp_apptrace_buffer_put(ESP_APTRACE_DEST_TRAX, ptr, 100/*tmo...
                                         in us");
if (res != ESP_OK) {
    /* in case of error host tracing tool (e.g. OpenOCD) will report...
     incomplete user buffer */
    ESP_LOGE(TAG, "Failed to put buffer!");
    return res;
}
```

另外，根据实际项目的需要，用户可能希望从主机接收数据。下面的代码片段展示了如何执行此操作。

```c
#include "esp_app_trace.h"
...
char buf[32];
char down_buf[32];
size_t sz = sizeof(buf);
/* config down buffer */
esp_apptrace_down_buffer_config(down_buf, sizeof(down_buf));
/* check for incoming data and read them if any */
```

(下页继续)
esp_err_t res = esp_apptrace_read(ESP_APPTRACE_DEST_TRAX, buf, &sz, 0/*do not wait*/);
if (res != ESP_OK) {
    ESP_LOGE(TAG, "Failed to read data from host!");
    return res;
} if (sz > 0) {
    /* we have data, process them */
    ...
}

esp_apptrace_read() 函数使用 memcpy 把主机端的数据复制到用户缓存区。在某些情况下，使用 esp_apptrace_down_buffer_get() 和 esp_apptrace_down_buffer_put() 函数可能更为理想。它们允许开发人员占用一块缓冲区并就地进行有关处理操作。下面的代码片段展示了如何执行此操作。

```c
#include "esp_app_trace.h"
...
char down_buf[32];
uint32_t *number;
size_t sz = 32;

/* config down buffer */
esp_apptrace_down_buffer_config(down_buf, sizeof(down_buf));
char *ptr = (char *)esp_apptrace_down_buffer_get(ESP_APPTRACE_DEST_TRAX, &sz, /*do not wait*/); 100/*tmo in us*/);
if (ptr == NULL) {  
    ESP_LOGE(TAG, "Failed to get buffer!");
    return ESP_FAIL;
} if (sz > 4) {
    number = (uint32_t *)ptr;
    printf("Here is the number %d", *number);
} else {
    printf("No data");
}
esp_err_t res = esp_apptrace_down_buffer_put(ESP_APPTRACE_DEST_TRAX, ptr, 100/*tmo in us*/);
if (res != ESP_OK) {
    /* in case of error host tracing tool (e.g. OpenOCD) will report incomplete user buffer */
    ESP_LOGE(TAG, "Failed to put buffer!");
    return res;
}
```

2. 下一步是编译应用程序的镜像，并将其下载到目标板上。这一步可以参考文档，构建并烧写。
3. 运行 OpenOCD（参见 JTAG 调试）。
4. 连接到 OpenOCD 的 telnet 服务器。用户可在终端执行命令 telnet <oocd_host> 4444。如果用户是在运行 OpenOCD 的同一台机器上打开 telnet 会话，可以使用 localhost 替换上面命令中的 <oocd_host>。
5. 使用特殊的 OpenOCD 命令开始收集待跟踪的命令。此命令将传输跟踪数据并将其重定向到指定的文件或套接字（当前仅支持文件作为跟踪数据目标）。相关命令的说明，请参阅 启动调试器。
6. 最后，处理接收到的数据。由于数据格式由用户自己定义，本文档中略数据处理的具体流程。数据处理的范例可以参考位于 $IDF_PATH/tools/esp_app_trace 下的 Python 脚本 apptrace_proc.py（用于功能测试）和 logtrace_proc.py（请参阅记录日志到主机 章节中的详细信息）。

**OpenOCD 应用程序跟踪命令** HW UP BUFFER 在用户数据块之间共享，并且会代替 API 调用者（在任务或者中断上下文中）填充分配到的内存。在多线程环境中，正在填充缓冲区的任务/中断可能会被另一个高优先级的任务/中断抢占，因此主机可能会读取到尚未准备好的用户数据。对此，跟踪模块在所有用户数据块之前添加一个数据头，其中包含有分配的用户缓冲区的大小（2 字节）和实际写入的数据长度。
(2 字节)，也就是说数据头总共 4 字节，负责读取跟踪数据的 OpenOCD 命令在读取到不完整的用户数据块时会报错。但是无论如何，它都会将整个用户数据块（包括还未填充的区域）的内容放到输出文件中。

下文介绍了如何使用 OpenOCD 应用程序跟踪命令。

备注：目前，OpenOCD 还不支持将任意用户数据发送到目标的命令。

命令用法：

```c
esp aptrace [start <options>] | [stop] | [status] | [dump <cores_num> <outfile>]
```

子命令：

- **start** 开始跟踪（连续流模式）。
- **stop** 停止跟踪。
- **status** 获取跟踪状态。
- **dump** 转储所有后验模式的数据。

Start 子命令的语法：

```c
start <outfile> [poll_period [trace_size [stop_tmo [wait4halt [skip_size]]]]]
```

- **outfile** 用于保存来自两个 CPU 的数据文件的路径。该参数需要具有以下格式：file://path/to/file。
- **poll_period** 轮询跟踪数据的周期（单位：毫秒）。如果大于 0 则以非阻塞模式运行。默认为 1 毫秒。
- **trace_size** 最多要收集的数据量（单位：字节），接收到指定数量的数据后将会停止跟踪。默认为 -1（禁用跟踪大小停止触发器）。
- **stop_tmo** 空闲超时（单位：秒）。如果指定的时间段内都没有数据就会停止跟踪。默认为 -1（禁用跟踪超时停止触发器）。还可以将其设置为以目标跟踪命令之间的最长暂停值更长的值（可选）。
- **wait4halt** 如果设置为 0 则立即开始跟踪。否则命令会先等待目标停止（复位、打断点等），然后对其进行自动恢复并开始跟踪。默认值为 0。
- **skip_size** 开始时要跳过的字节数，默认为 0。

备注：如果 poll_period 为 0，则在跟踪停止之前，OpenOCD 的 telnet 命令将不可用。必须通过复位电路板或者在 OpenOCD 的窗口中（非 telnet 会话窗口）使用快捷键 Ctrl+C。另一种选择是设置 trace_size 并等待，当收集到指定数据量时，跟踪会自动停止。

命令使用示例：

1. 将 2048 个字节的跟踪数据收集到 trace.log 文件中。该文件将保存在 openocd-esp32 目录中。

   ```c
   esp aptrace start file://trace.log 1 2048 5 0 0
   ```

   跟踪数据会被检查并以非阻塞的模式保存到文件中。如果收集满 2048 字节的数据或者在 5 秒内没有新的数据，那么该过程就会停止。

   备注：在将数据提供给 OpenOCD 之前，会对其进行缓冲。如果看到“Data timeout!”的消息，这表示目标可能在超时之前没有向 OpenOCD 发送足够的数据以清空缓冲区。要解决这个问题，可以增加超时时间或者使用函数 esp_aptrace_flush() 以特定间隔刷新数据。

2. 在非阻塞模式下无限地检索跟踪数据。

   ```c
   esp aptrace start file://trace.log 1 -1 -1 0 0
   ```

   对收集数据的大小没有限制，也不设置超时时间。要停止此过程，可以在 OpenOCD 的 telnet 会话窗口中发送 esp_aptrace stop 命令，或者在 OpenOCD 窗口中使用快捷键 Ctrl+C。

3. 检索跟踪数据并无限期保存。
记录日志到主机

记录日志到主机是 ESP-IDF 中一个非常实用的功能：通过应用层跟踪库将日志保存到主机端。某种程度上，这也算是一种半主机 (semihosting) 机制，相较于调用 ESP_LOGx 将待打印的字符串发送到 UART 的日志记录方式，此功能将大部分工作转移到了主机端，从而减少了本地工作量。

ESP-IDF的日志库会默认使用 vprintf 的函数将格式化的字符串输出到专用的 UART，一般来说涉及以下几个步骤：

1. 解析格式字符串以获取每个参数的类型。
2. 根据其类型，将每个参数都转换为字符串。
3. 格式字符串与转换后的参数一起发送到 UART。

虽然可以对类 vprintf 函数进行一定程度的优化，但由于在任何情况下都必须执行上述步骤，并且每个步骤都会消耗一定的时间（尤其是步骤 3），所以经常会发生以下这种情况：向程序中添加额外的打印信息以诊断问题，却改变了应用程序的行为，使得问题无法复现。在最严重的情况下，程序无法正常工作，最终导致报错甚至挂起。

想要解决此类问题，可以使用更高的波特率或者其他更快的接口，并将字符串格式化的工作转移到主机端。

通过应用层跟踪库的 esp_aptrace_vprintf 函数，可以将日志信息发送到主机。该函数不执行格式字符串的参数的完全解析，而仅仅计算传递参数的数量，并将它们与格式字符串地址一起发送给主机。主机端会通过一个特殊的 Python 脚本来处理并打印接收到的日志数据。

局限  目前通过 JTAG 实现记录日志还存在以下几点局限：

1. 不支持使用 ESP_EARLY_LOGx 宏进行跟踪。
2. 不支持大于超过 4 字节的 printf 参数（例如 double 和 uint64_t）。
3. 仅支持.rodata 段中的格式字符串和参数。
4. 最多支持 256 个 printf 参数。

如何使用  为了使用跟踪模块来记录日志，用户需要执行以下步骤：

1. 在目标端，需要安装特殊的类 vprintf 函数 esp_aptrace_vprintf，该函数负责将日志数据发送给主机。示例代码参见 system/app_trace_to_host。
2. 按照特定应用程序的跟踪 章节中的第 2-5 步进行操作。
3. 打印接收到的日志记录，请在终端运行以下命令：$IDF_PATH/tools/esp_app_trace/logtrace_proc.py /path/to/trace/file /path/to/program/elf/file。

Log Trace Processor 命令选项  命令用法：

logtrace_proc.py [-h] [--no-errors] <trace_file> <elf_file>

位置参数（必要）：

.trace_file 日志跟踪文件的路径。
.elf_file 程序 ELF 文件的路径。

可选参数：

-h, --help 显示此帮助信息并退出。
--no-errors, -n 不打印错误信息。

基于 SEGGER SystemView 的系统行为分析

ESP-IDF 中另一个基于应用层跟踪库的实用功能是系统级跟踪，它会生成与 SEGGER SystemView 工具相兼容的跟踪信息。SEGGER SystemView 是一款实时记录和可视化工具，用来分析应用程序运行时的行为，可通过 UART 接口实时查看事件。

如何使用 若需使用这个功能，需要在 menuconfig 中开启 CONFIG_APPTRACE_SV_ENABLE 选项，具体路径为 Component config > Application Level Tracing > FreeRTOS SystemView Tracing。同一菜单栏下还开启了其它几个选项：

1. System View destination，选择需要使用的接口：JTAG 或 UART。使用 UART 接口时，可以将 SystemView 应用程序直接连接到 ESP32 并实时接收数据。
2. ESP32 timer to use as SystemView timestamp source（CONFIG_APPTRACE_SV_TS_SOURCE）。选择 SystemView 事件使用的时间戳源。在单核模式下，使用 ESP32 内部的循环计数器生成时间戳，其最大的工作频率是 240 MHz（时间戳粒度大约为 4 ns）。在双核模式下，使用工作在 40 MHz 的外部定时器，因此时间戳粒度为 25 ns。
3. 可以单独启用或禁用的 SystemView 事件集合（CONFIG_APPTRACE_SV_EVT_XXX）：
   - Trace Buffer Overflow Event
   - ISR Enter Event
   - ISR Exit Event
   - ISR Exit to Scheduler Event
   - Task Start Execution Event
   - Task Stop Execution Event
   - Task Start Ready State Event
   - Task Stop Ready State Event
   - Task Create Event
   - Task Terminate Event
   - System Idle Event
   - Timer Enter Event
   - Timer Exit Event

ESP-IDF 中已经包含了所有用于生成兼容 SystemView 跟踪信息的代码，用户只需配置必要的项目选项（如上所示），然后构建、烧写映像到目标板，接着参照前面的介绍，使用 OpenOCD 收集数据。

4. 想要通过 UART 接口进行实时跟踪，请在菜单配置选项 Component config > Application Level Tracing > FreeRTOS SystemView Tracing 中选择 Pro 或 App CPU。

OpenOCD SystemView 跟踪命令选项 命令用法：

esp sysview [start <options>] | [stop] | [status]

子命令：

start 开启跟踪（连续流模式）。
stop 停止跟踪。
status 获取跟踪状态。

Start 子命令语法：

start <ofile1> [ofile2] [poll_period [trace_size [stop_tmo]]]

ofile1 保存 PRO CPU 数据的文件路径。此参数需要具有如下格式：file://path/to/file。
ofile2 保存 APP CPU 数据的文件路径。此参数需要具有如下格式：file://path/to/file。
poll_period 跟踪数据的轮询周期（单位：毫秒）。如果该值大于 0，则命令以非阻塞的模式运行，默认为 1 毫秒。
trace_size 最多要收集的数据量（单位：字节）。当收到指定数量的数据后，将停止跟踪。默认值是 -1 （禁用跟踪大小停止触发器）。
Chapter 4. API 指南

**stop_tmo** 空闲超时 (单位: 秒)。如果指定的时间内没有数据，将停止跟踪。默认值是 -1（禁用跟踪超时停止触发器）。

**备注:** 如果 poll_period 为 0，则在跟踪停止之前，OpenOCD 的 telnet 命令行将不可用。您需要复位板卡或者在 OpenOCD 的窗口（非 telnet 会话窗口）输入 Ctrl+C 命令来手动停止跟踪。另一个办法是设置 trace_size，等到收集满指定数量的数据后自动停止跟踪。

命令使用示例:
1. 将 SystemView 跟踪数据收集到文件 pro-cpu.SVDat 和 pro-cpu.SVDat 中。这些文件会被保存在 openocd-esp32 目录中。

   ```
   esp sysview start file://pro-cpu.SVDat file://app-cpu.SVDat
   ```

   跟踪数据被检索并以非阻塞的方式保存。要停止此过程，需要在 OpenOCD 的 telnet 会话窗口输入 esp sysview stop 命令，也可以在 OpenOCD 窗口中按下快捷键 Ctrl+C。

2. 检索跟踪数据并无限保存。

   ```
   esp32 sysview start file://pro-cpu.SVDat file://app-cpu.SVDat 0 -1 -1
   ```

   OpenOCD 的 telnet 命令行在跟踪停止前会无法使用。要停止跟踪，请在 OpenOCD 窗口使用 Ctrl+C 快捷键。

**数据可视化** 收集到跟踪数据后，用户可以使用特殊的工具对结果进行可视化并分析程序行为。

遗憾的是，SystemView 不支持从多个核心进行跟踪。所以当使用 JTAG 追踪双核模式下的 ESP32 时会生成两个文件：一个用于 PRO CPU，另一个用于 APP CPU。用户可以将每个文件加载到工具中单独分析。使用 UART 进行追踪时，可以在 menuconfig Pro 或 App 中点击 Component config > Application Level Tracing > FreeRTOS SystemView Tracing 并选择要追踪的 CPU。

在工具中单独分析每个核的跟踪数据是比较棘手的，但是 Eclipse 提供了 Impulse 插件，该插件可以加载多个跟踪文件，并且可以在同一视图中检查来自两个内核的事件。此外，与免费版的 SystemView 相比，此插件没有1,000,000个事件的限制。

关于如何安装、配置 Impulse 并使用它来可视化来自单个核心的跟踪数据，请参阅 [官方教程](#)。

**备注:** ESP-IDF 使用自己的 SystemView FreeRTOS 事件 ID 映射，因此用户需要将 SYSVIEW_INSTALL_DIR/Description/SYSVIEW_FreeRTOS.txt 替换成 $IDF_PATH/docs/api-guides/SYSVIEW_FreeRTOS.txt。在使用上述链接配置 SystemView 序列化程序时，也应该使用该特定文件的内容。

**配置 Impulse 实现双核跟踪** 在安装好 Impulse 插件并确保 Impulse 能够在单独的选项卡中成功加载每个核心的跟踪文件后，用户可以添加特殊的 Multi Adapter 端口并将这两个文件加载到一个视图中。为此，用户需要在 Eclipse 中执行以下操作：

1. 打开 Signal Ports 视图，前往 Windows > Show View > Other 菜单，在 Impulse 文件夹中找到 Signal Ports 视图并双击。
2. 在 Signal Ports 视图中，右键 Ports 并选择 Add，然后选择 New Multi Adapter Port。
3. 在打开的对话框中按下 add 按钮，选择 New Pipe/File。
4. 在打开的对话框中选择 SystemView Serializer 并设置 PRO CPU 跟踪文件的路径，按下 OK 保存设置。
5. 对 APP CPU 的跟踪文件重复步骤 3 和 4。
6. 双击创建的端口，会打开此端口的视图。
7. 单击 Start/Stop Streaming 按钮，数据将被加载。
8. 使用 Zoom Out，Zoom In 和 Zoom Fit 按钮来查看数据。
9. 有关设置测量光标和其他的功能，请参阅 Impulse官方文档。
Chapter 4. API 指南

备注：如果您在可视化方面遇到了问题（未显示数据或者缩放操作异常），您可以尝试删除当前的信号层次结构，再双击必要的文件或端口。Eclipse 会请求您创建新的信号层次结构。

Gcov（源代码覆盖）

Gcov 和 Gcovr 简介 源代码覆盖率显示程序运行时间内执行的每一条程序执行路径的数量和频率。Gcov 是一款 GCC 工具，与编译器协同使用时，可生成日志文件，显示源文件每行的执行次数。Gcovr 是管理 Gcov 和生成代码覆盖率总结的工具。

一般来说，使用 Gcov 在主机上编译和运行程序会经过以下步骤：

1. 使用 GCC 及 --coverage 选项编译源代码。编译器在编译过程中生成一个 gcno 注释文件，该文件包含逻辑执行路径块图以及将每个块映射到源代码行号等信息。每个用 --coverage 选项编译的源文件都会生成自己的同名 gcno 文件（如 main.c 在编译时会生成 main.gcno)。
2. 执行程序。在执行过程中，程序会生成 gcda 数据文件。这些数据文件包含了执行路径的次数统计。程序将为每个用 --coverage 选项编译的源文件生成一个 gcda 文件（如 main.c 将生成 main.gcda)。
3. Gcov 或 Gcovr 可用于生成基于 gcno、.gcda 和源文件的代码覆盖。Gcov 将以 gcov 文件的形式为每个源文件生成基于文本的覆盖报告，而 Gcovr 将以 HTML 格式生成覆盖报告。

ESP-IDF 中的 Gcov 和 Gcovr 应用 在 ESP-IDF 中使用 Gcov 的过程比较复杂，因为程序不在主机上运行，而在目标机上运行。代码覆盖率数据（即 gcda 文件）最初存储在目标机上，OpenOCD 在运行时通过 JTAG 将代码覆盖率数据从目标机传输到主机上。在 ESP-IDF 中使用 Gcov 可以分为以下几个步骤：

1. 为 Gcov 设置项目
2. 转储代码覆盖数据
3. 生成代码覆盖报告

为 Gcov 设置项目

编译器选择 为了获取项目的代码覆盖率数据，必须用 --coverage 选项编译项目中的一个或多个源文件。在 ESP-IDF 中，这可以在组件级或单个源文件级实现：

- 在组件的 CMakeLists.txt 文件中添加 target_compile_options("${COMPONENT_LIB} PRIVATE --coverage") 可确保使用 --coverage 选项编译组件中的所有源文件。
- 在组件的 CMakeLists.txt 文件中添加 set_source_files_properties(source1.c source2.c PROPERTIES COMPILE_FLAGS --coverage) 可确保使用 --coverage 选项编译同一组件中选定的一源文件（如 source1.c 和 source2.c)。

当一个源文件用 --coverage 选项编译时（例如 gcov_example.c)，编译器会在项目的构建目录下生成 gcov_example.gcno 文件。

项目配置 在构建有源代码覆盖的项目之前，请运行 idf.py menuconfig 以启用以下项目配置选项。

- 通过 CONFIG_APPTRACE_DESTINATION1 选项选择 Trace Memory 来启用应用程序跟踪模块。
- 通过 CONFIG_APPTRACE_GCOV_ENABLE 选项启用 Gcov 主机。

转储代码覆盖数据 一旦项目使用 --coverage 选项编译并烧录到目标机上，应用程序运行时，代码覆盖率数据将存储在目标机内部（即在跟踪存储器中）。将代码覆盖率数据从目标机转移到主机上的过程称为转储。

覆盖率数据的转储通过 OpenOCD 进行（关于如何设置和运行 OpenOCD，请参考 JTAG 调试）。由于该过程需要通过向 OpenOCD 发出命令来触发转储，因此必须打开 telnet 会话，以向 OpenOCD 发出这些命令（运行 telnet localhost 4444)，GDB 也可以代替 telnet 来向 OpenOCD 发出命令，但是所有从 GDB 发出的命令都需要以 mon <oocd_command> 为前缀。

Submit Document Feedback
当目标机转储代码覆盖数据时,.gcda 文件存储在项目的构建目录中。例如，如果 main 组件的 gcov_example_main.c 在编译时使用了--coverage 选项，那么转储代码覆盖数据将在 build/esp-idf/main/CMakeFiles/__idf_main.dir/gcov_example_main.c.gcda 中生成 gcov_example_main.gcda 文件。注意，编译过程中产生的.gcno 文件也放在同一目录下。

代码覆盖数据的转储可以在应用程序的整个生命周期内多次进行。每次转储都会用最新的代码覆盖信息更新 .gcda 文件，代码覆盖数据是累积的，因此最新的数据将包含应用程序整个生命周期中每个代码路径的总执行次数。

ESP-IDF 支持两种将代码覆盖数据从目标机转储到主机的方法：

- 运行中实时转储
- 硬编码转储

运行中实时转储 通过 telnet 会话调用 OpenOCD 命令 ESP32 gcov 来触发运行时的实时转储。一旦被调用，OpenOCD 将立即抢占 ESP32 的当前状态，并执行内置的 ESP-IDF Gcov 调试存根函数。调试存根函数将数据转储到主机。完成后，ESP32 将恢复当前状态。

硬编码转储 硬编码转储是由应用程序本身从程序内部调用 esp_gcov_dump() 函数触发的。在调用时，应用程序将停止并等待 OpenOCD 连接，同时检索代码覆盖数据。一旦 esp_gcov_dump() 函数被调用，主机会通过 telnet 会话执行 esp_gcov_dump OpenOCD 命令。该命令会将 OpenOCD 连接到 ESP32，检索代码覆盖数据，然后断开与 ESP32 的连接，从而恢复应用程序。在应用程序的生命周期中可多次触发硬编码转储。

在必要时（如应用程序初始化后或是应用程序主循环的每次迭代期间）放置 esp_gcov_dump()，当应用程序在生命周期的某刻需要代码覆盖率数据时，硬编码转储会非常有用。

GDB 可以用来在 esp_gcov_dump() 上设置断点，然后使用 gdbinit 脚本自动调用 mon esp gcov dump（关于 GDB 的使用可参考使用命令行调试）。

以下 GDB 脚本将在 esp_gcov_dump() 处添加一个断点，然后调用 mon esp gcov dump OpenOCD 命令。

```bash
b esp_gcov_dump
commands
mon esp gcov dump
end
```

备注：注意，所有的 OpenOCD 命令都应该在 GDB 中以 mon <oocd_command> 方式调用。

生成代码覆盖报告 一旦代码覆盖数据被转储，.gcno、.gcda 和源文件可以用来生成代码覆盖报告。该报告会显示源文件中每行被执行的次数。

Gcov 和 Gcovr 都可以用来生成代码覆盖报告。安装 Xtensa 工具链时会一起安装 Gcov，但 Gcovr 可能需要单独安装。关于如何使用 Gcov 或 Gcovr，请参考 Gcov 文档 和 Gcovr 文档。

在工程中添加 Gcov 构建目标 用户可以在自己的工程中定义额外的构建目标，从而通过一个简单的构建命令即可更方便地生成报告。

请在您工程的 CMakeLists.txt 文件中添加以下内容：

```cmake
include(${ENV{IDF_PATH}/tools/cmake/gcov.cmake)
idf_create_coverage_report(${CMAKE_CURRENT_BINARY_DIR}/coverage_report)
idf_clean_coverage_report(${CMAKE_CURRENT_BINARY_DIR}/coverage_report)
```

您可使用以下命令：

- `cmake --build build/ --target gcovr-report`: 在 $(BUILD_DIR_BASE)/coverage_report/html 目录下生成 HTML 格式代码覆盖报告。

Espressif Systems 2116 Release v5.1-dev-2066-g7869f4e151
4.2 应用程序的启动流程

本文将会介绍 ESP32 从上电到运行 app_main 函数中所经历的步骤 (即启动流程)。

宏观上，该启动流程可以分为如下 3 个步骤:

1. **一级引导程序** 被固化在了 ESP32 内部的 ROM 中，它会从 flash 的 0x1000 偏移地址处加载二级引导程序至 RAM (IRAM & DRAM) 中。
2. **二级引导程序** 从 flash 中加载分区表和主程序镜像至内存中，主程序中包含了 RAM 段和通过 flash 高速缓存映射的只读段。
3. **应用程序启动阶段** 运行，这时第二个 CPU 和 RTOS 的调度器启动。

下面会对上述过程进行更为详细的阐述。

### 4.2.1 一级引导程序

SoC 复位后，PRO CPU 会立即开始运行，执行复位向量代码，而 APP CPU 仍然保持复位状态。在启动过程中，PRO CPU 会执行所有的初始化操作。APP CPU 的复位状态会在应用程序启动代码的 call_start_cpu0 函数中失效。复位向量代码位于 ESP32 芯片映射 ROM 处，且不能被修改。

复位向量调用的启动代码会根据 GPIO_STRAP_REG 寄存器的值来确定 ESP32 的启动模式，该寄存器保存着复位后 bootstrap 引脚的电平状态。根据不同的复位原因，程序会执行如下操作:

1. **从深度睡眠模式复位**：如 RTC_CNTL_STORE6_REG 寄存器的值非零，且 RTC_CNTL_STORE7_REG 寄存器中的 RTC 内存的 CRC 校验值有效，那么程序会使用 RTC_CNTL_STORE6_REG 寄存器的值作为入口地址，并立即跳转到该地址运行。如果 RTC_CNTL_STORE6_REG 的值为零，或 RTC_CNTL_STORE7_REG 中的 CRC 校验值无效，又或通过 RTC_CNTL_STORE6_REG 调用的代码返回，那么则像上电复位一样继续启动。

**注意**：如果想在这里运行自定义的代码，可以参考深度睡眠文档里面介绍的深度睡眠存根机制方法。

2. **上电复位**，软件 SoC 复位。看门狗 SoC 复位：检查 GPIO Strap 寄存器，判断是否请求自定义启动模式，如 UART 下载模式。如果是，ROM 会执行此自定义加载器模式。否则程序将从软件 CPU 复位一样继续启动。请参考 ESP32 技术规格书了解 SoC 启动模式以及具体执行过程。

3. **软件 CPU 复位**，看门狗 CPU 复位：根据 EFUSE 中的值配置 SPI flash，然后尝试从 flash 中加载代码，这部分将会在后面一小节详细介绍。

**备注**：正常启动模式下会使能 RTC 看门狗，因此，如果进程中断或停止，看门狗将自动重置 SOC 并重启启动过程。如果 strapping GPIOs 已更改，则可能导致 SoC 陷入新的启动模式。

二级引导程序的二进制镜像会从 flash 的 0x1000 偏移地址处加载。如果正在使用 Secure Boot，则 flash 的第一个 4 kB 扇区用于存储安全启动 IV 以及引导程序镜像的摘要，否则不使用该扇区。

### 4.2.2 二级引导程序

在 ESP-IDF 中，存放在 flash 的 0x1000 偏移地址处的二进制镜像就是二级引导程序。二级引导程序的源码可以在 ESP-IDF 的 components.bootloader 目录下找到。ESP-IDF 使用二级引导程序可以增加 flash 分区的灵活性（使用分区表），并且方便实现 flash 加密，安全引导和空中升级（OTA）等功能。

当一级引导程序校验并加载完二级引导程序后，它会从二进制镜像的头部找到二级引导程序的入口点，并跳转过去运行。
4.2.3 应用程序启动阶段

应用程序启动包含了从应用程序开始执行到 app_main 函数在主任务内部运行前的所有过程。可分为三个阶段:

- 硬件和基本 C 语言运行环境的端口初始化。
- 软件服务和 FreeRTOS 的系统初始化。
- 运行主任务并调用 app_main。

备注：通常不需要了解 ESP-IDF 应用程序初始化的所有阶段。如果需要仅从应用程序开发人员的角度了解初始化，请跳至运行主任务。

端口初始化

ESP-IDF 应用程序的入口是 components/esp_system/port/cpu_start.c 文件中的 call_start_cpu0 函数。这个函数由二级引导加载程序执行，并且从不返回。

该端口层的初始化功能会初始化基本的 C 运行环境（“CRT”），并对 SoC 的内部硬件进行了初始化。

- 为应用程序重新配置 CPU 异常（允许应用程序中断处理程序运行，并使用为应用程序配置的选项来处理严重错误，而不是使用 ROM 提供的简易版错误处理程序处理）。
- 如果没有设置选项 CONFIG_BOOTLOADER_WDT_DISABLE，则不使用 RTC 看门狗定时器。
- 初始化内部存储器（数据和 bss）。
- 完成 MMU 高速缓存配置。
- 如果配置了 PSRAM，则使能 PSRAM。
- 将 CPU 时钟设置为配置项的频率。
- 根据应用程序头部配置重新配置主 SPI flash，这是为了与 ESP-IDF V4.0 之前的引导程序版本兼容，请参考引导加载程序兼容性。
- 如果应用程序被配置为在多个内核上运行，则启动另一个内核并等待其初始化（在类似的“端口层”初始化函数 call_start_cpu1 内）。

call_start_cpu0 完成运行后，将会调用在 components/esp_system/startup.c 中找到的“系统层”初始化函数 start_cpu0。其他内核也将完成端口层的初始化，并调用同一文件中的 start_other_cores。
系统初始化

主要的系统初始化函数是 start_cpu0。默认情况下，这个函数与 start_cpu0_default 函数弱连接。这意味着可以覆盖这个函数，增加一些额外的初始化步骤。

主要的系统初始化阶段包括:

- 如果默认的目录级别允许，则记录该应用程序的相关信息（项目名称、应用程序版本等）。
- 初始化堆分配器（在执行之前，所有分配必须是静态的或在堆栈上）。
- 初始化 newlib 组件的系统调用和时间函数。
- 配置断电检测器。
- 根据串行控制台配置设置 libc stdin、stdout、和 stderr。
- 执行与安全有关的检查，包括为该配置烧录 fuse（包括禁用 ESP32 V3 的 ROM 下载模式，CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE）。
- 初始化 SPI flash API 支持。
- 调用全局 C++ 枚举函数和任何标有__attribute__((constructor)) 的 C 函数。

二级系统初始化允许单个组件被初始化。如果一个组件有一个用 ESP_SYSTEM_INIT_FN 宏注释的初始化函数，它将作为二级初始化的一部分被调用。

运行主任务

在所有其他组件都初始化后，主任务会被创建。FreeRTOS 调度器开始运行。

做完一些初始化任务后（需要启动调度器），主任务在固件中运行应用程序提供的函数 app_main。

运行 app_main 的主任务有一个固定的 RTOS 优先级（比最小值高）和一个可配置的堆栈大小。

主任务的内核亲和性也是可以配置的，请参考 CONFIG_ESP_MAIN_TASK_AFFINITY。

与普通的 FreeRTOS 任务（或嵌入式 C 的 main 函数）不同，app_main 任务可以返回。如果“app_main”函数返回，那么主任务将被删除。系统将继续运行其他 RTOS 任务。因此可以将 app_main 实现为一个创建其他应用任务然后返回的函数，或主应用任务本身。

APP CPU 的内核启动流程

APP CPU 的启动流程类似但更简单:

当运行系统初始化时，PRO CPU 上的代码会给 APP CPU 设置好入口地址，解除其复位状态，然后等待 APP CPU 上运行的代码设置一个全局标志，以表明 APP CPU 已经正常启动。完成后，APP CPU 跳转到 components/esp_system/port/cpu_start.c 中的 call_start_cpu1 函数。

当 start_cpu0 函数对 PRO CPU 进行初始化的时候，APP CPU 运行 start_cpu_other_cores 函数。与 start_cpu0 函数类似，start_cpu_other_cores 函数是弱链接的，默认为 start_cpu_other_cores_default 函数，但可以由应用程序替换为不同的函数。

start_cpu_other_cores_default 函数做了一些与内核相关的系统初始化，然后等待 PRO CPU 启动 FreeRTOS 的调度器。启动完成后，它会执行 esp_startup_start_app_other_cores 函数，这是一个默认为 esp_startup_start_app_other_cores_default 的弱链接函数。

默认情况下，esp_startup_start_app_other_cores_default 只会自旋，直到 PRO CPU 上的调度器触发中断，以启动 APP CPU 上的 RTOS 调度器。

4.3 BluFi
4.3.1 概览

BluFi 是一款基于蓝牙通道的 Wi-Fi 网络配置功能，适用于 ESP32。它通过安全协议将 Wi-Fi 的 SSID、密码等配置信息传输到 ESP32。基于这些信息，ESP32 可进而连接到 AP 或建立 SoftAP。

BluFi 流程的关键部分包括数据的分片、加密以及校验和验证。用户可按需自定义用于对称加密、非对称加密以及校验的算法。此处，我们采用 DH 算法进行密钥协商，128-AES 算法用于数据加密，CRC16 算法用于校验和验证。

4.3.2 BluFi 流程

BluFi 配网流程包含配置 SoftAP 和配置 Station 两部分。

下面以配置 Station 为例，介绍了广播、连接、服务发现、协商共享密钥、传输数据、回传连接状态等关键步骤。

1. ESP32 开启 GATT Server 模式，发送带有特定 advertising data 的广播。该广播不属于 BluFi Profile，您可以按需对其进行自定义。
2. 使用手机应用程序搜索到该广播后，手机将作为 GATT Client 连接 ESP32。该步骤对具体使用哪款手机应用程序并无特殊要求。
3. 成功建立 GATT 连接后，手机会向 ESP32 发送数据帧进行密钥协商（详见 BluFi 中定义的帧格式）。
4. ESP32 收到密钥协商的数据帧后，会按照自定义的协商方法进行解析。
5. 手机与 ESP32 进行密钥协商。协商过程可使用 DH/RSA/ECC 等加密算法。
6. 协商结束后，手机将向 ESP32 发送控制帧，用于设置安全模式。
7. ESP32 收到控制帧后，使用共享密钥以及安全配置对通信数据进行加密和解密。
8. 手机向 ESP32 发送 BluFi 中定义的帧格式 中定义的数据帧，包括 SSID、密码等 Wi-Fi 配置信息。
9. 手机向 ESP32 发送 Wi-Fi 连接请求的控制帧。ESP32 收到控制帧后，即默认手机已完成必要信息的传输，准备连接 Wi-Fi。
10. 连接到 Wi-Fi 后，ESP32 发送 Wi-Fi 连接状态报告的控制帧到手机。至此，配网结束。

备注：

1. ESP32 收到安全模式配置的控制帧后，会根据定义的安全模式进行相关操作。
2. 进行对称加密和解密时，加密和解密前后的数据长度必须一致。支持原地加密和解密。

4.3.3 BluFi 流程图

4.3.4 BluFi 中定义的帧格式

手机应用程序与 ESP32 之间的 BluFi 通信格式定义如下：

帧不分片格式：

<table>
<thead>
<tr>
<th>字段</th>
<th>值（字节）</th>
</tr>
</thead>
<tbody>
<tr>
<td>类型（最低有效位）</td>
<td>1</td>
</tr>
<tr>
<td>帧控制</td>
<td>1</td>
</tr>
<tr>
<td>序列号</td>
<td>1</td>
</tr>
<tr>
<td>数据长度</td>
<td>1</td>
</tr>
<tr>
<td>数据</td>
<td>$[\text{Data Length}]$</td>
</tr>
<tr>
<td>校验（最高有效位）</td>
<td>2</td>
</tr>
</tbody>
</table>

如果使用 帧控制 字段中的分片位，则 数据 字段中会出现 2 字节的 内容总长度。该 内容总长度 表示帧的剩余部分的总长度，并用于报告终端需要分配的内存大小。

帧分片格式：

Espressif Systems 2120 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
图 2: BluFi Flow Chart
## 字段

<table>
<thead>
<tr>
<th>字段</th>
<th>值（字节）</th>
</tr>
</thead>
<tbody>
<tr>
<td>类型（最低有效位）</td>
<td>1</td>
</tr>
<tr>
<td>帧控制（分片）</td>
<td>1</td>
</tr>
<tr>
<td>序列号</td>
<td>1</td>
</tr>
<tr>
<td>数据长度</td>
<td>1</td>
</tr>
<tr>
<td>数据</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 内容总长度：2</td>
</tr>
<tr>
<td></td>
<td>• 数据内容长度：${\text{Data Length}} - 2$</td>
</tr>
<tr>
<td>校验（最高有效位）</td>
<td>2</td>
</tr>
</tbody>
</table>

通常情况下，控制帧不包含数据位，ACK 帧类型除外。

**ACK 帧格式（8 bit）：**

<table>
<thead>
<tr>
<th>字段</th>
<th>值（字节）</th>
</tr>
</thead>
<tbody>
<tr>
<td>类型 - ACK（最低有效位）</td>
<td>1</td>
</tr>
<tr>
<td>帧控制</td>
<td>1</td>
</tr>
<tr>
<td>序列号</td>
<td>1</td>
</tr>
<tr>
<td>数据长度</td>
<td>1</td>
</tr>
<tr>
<td>数据</td>
<td>ACK 序列号: 2</td>
</tr>
<tr>
<td>校验（最高有效位）</td>
<td>2</td>
</tr>
</tbody>
</table>

1. **类型字段**
   - 类型字段占 1 字节，分为 类型 和 子类型 两部分。其中，类型占低 2 位，表明该帧为数据帧或是控制帧；子类型占高 6 位，表示此数据帧或者控制帧的具体含义。
     - 控制帧，暂不进行加密，可校验。
     - 数据帧，可加密，可校验。

1.1 **控制帧（二进制：0x0 b’00）**
### Chapter 4. API 指南

<table>
<thead>
<tr>
<th>控制帧</th>
<th>含义</th>
<th>解释</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0 (b'000000)</td>
<td>ACK</td>
<td>ACK 帧的数据字段使用回复对象帧的数据值。</td>
<td>数据字段占用 1 字节，其序列值与回复对象帧的数据值相同。</td>
</tr>
<tr>
<td>0x1 (b'000001)</td>
<td>将 ESP 设备设置为安全模式。</td>
<td>通知 ESP 设备发送数据时使用的安全模式，在数据发送过程中多次重置，设置后将影响后续使用的安全模式。</td>
<td>数据字段占用 1 字节。高 4 位用于控制帧的安全模式设置，低 4 位用于数据帧的安全模式设置。</td>
</tr>
</tbody>
</table>
| 0x2 (b'000010) | Wi-Fi 的 opmode。 | 该帧包含设置 ESP 设备 Wi-Fi 模式 (opmode) 的设置信息。 | data[0] 用于设置 opmode，包括：
- 0x00: NULL
- 0x01: STA
- 0x02: SoftAP
- 0x03: SoftAP & STA
如果设置中包含 AP，请尽量优先设置 AP 模式的 SSID/密码/最大连接数等。 |
| 0x3 (b'000011) | 将 ESP 设备连接至 AP。 | 通知 ESP 设备必要的信息已经发送完毕，可以连接至 AP。 | 不包含数据字段。 |
| 0x4 (b'000100) | 断开 ESP 设备与 AP 的连接。 | | 不包含数据字段。 |
| 0x5 (b'000101) | 获取 ESP 设备的 Wi-Fi 模式和状态等信息。 | | 不包含数据字段。ESP 设备收到此控制帧后，会向手机发送一个报告 Wi-Fi 连接状态的帧来告知手机当前所处的 opmode、连接状态、SSID 等信息。
- 提供给手机端的信息类型由手机上的应用程序决定。 |
| 0x6 (b'000110) | 断开 STA 设备与 SoftAP 的连接。 | data[0-5] 为 STA 设备的 MAC 地址。如有多个 STA 设备，则第二个使用 data[6-11]，依次类推。 |
| 0x7 (b'000111) | 获取版本信息。 | | |
| 0x8 (b'001000) | 断开 BLE GATT 连接。 | ESP 设备收到该指令后主动断开 BLE GATT 连接。 |
| 0x9 (b'001001) | 获取 Wi-Fi 列表。 | 通知 ESP 设备扫描周围的 Wi-Fi 热点。 | 不包含数据字段。ESP 设备收到此控制帧后，会向手机发送一个包含 Wi-Fi 热点报告的帧。 |

1.2 数据帧 (二进制：0x1 b’01)
<table>
<thead>
<tr>
<th>数据帧</th>
<th>含义</th>
<th>解释</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 (b'000000')</td>
<td>发送协商数据。</td>
<td>协商数据会发送到应用层注册的回调函数中。</td>
<td>数据的长度取决于数据长度字段。</td>
</tr>
<tr>
<td>0x01 (b'000001')</td>
<td>发送 STA 模式的 BSSID。</td>
<td>在 SSID 隐藏的情况下，发送 STA 设备要连接的 AP 的 BSSID。</td>
<td>请参考备注 1。</td>
</tr>
<tr>
<td>0x02 (b'000010')</td>
<td>发送 STA 模式的 SSID。</td>
<td>发送 STA 设备要连接的 AP 的 SSID。</td>
<td>请参考备注 1。</td>
</tr>
<tr>
<td>0x03 (b'000011')</td>
<td>发送 STA 模式的密码。</td>
<td>发送 STA 设备要连接的 AP 的密码。</td>
<td>请参考备注 1。</td>
</tr>
<tr>
<td>0x04 (b'000100')</td>
<td>发送 SoftAP 模式的 SSID。</td>
<td>请参考备注 1。</td>
<td></td>
</tr>
<tr>
<td>0x05 (b'000101')</td>
<td>发送 SoftAP 模式的密码。</td>
<td>请参考备注 1。</td>
<td></td>
</tr>
<tr>
<td>0x06 (b'000110')</td>
<td>设置 SoftAP 模式的最大连接数。</td>
<td>data[0] 为连接数的值，范围从 1 到 4。当传输方向是 ESP 设备到手机时，表示向手机端提供所需信息。</td>
<td></td>
</tr>
</tbody>
</table>
| 0x07 (b'000111') | 设置 SoftAP 的认证模式。 | data[0] 包括：
  - 0x00: OPEN
  - 0x01: WEP
  - 0x02: WPA_PSK
  - 0x03: WPA2_PSK
  - 0x04: WPA_WPA2_PSK | 请参考备注 1。 |
| 0x08 (b'001000') | 设置 SoftAP 模式的通道数量。 | data[0] 代表支持的通道的数量，范围从 1 到 14。若传输方向是从 ESP 设备到手机，则表示向手机端提供所需信息。 |
| 0x09 (b'001001') | 用户名 | 在进行企业级加密时提供 GATT 客户端的用户名。 | 数据的长度取决于数据长度字段。 |
| 0x0a (b'001010') | CA 认证 | 在进行企业级加密时提供 CA 认证。 | 请参考备注 2。 |
| 0x0b (b'001011') | 客户端认证 | 在进行企业级加密时提供客户端认证。是否包含私钥，取决于认证的内容。 | 请参考备注 2。 |
| 0x0c (b'001100') | 服务端认证 | 在进行企业级加密时提供服务端认证。是否包含私钥，取决于认证的内容。 | 请参考备注 2。 |
| 0x0d (b'001101') | 客户端私钥 | 在进行企业级加密时提供客户端私钥。 | 请参考备注 2。 |
| 0xe (b'001110') | 服务端私钥 | 在进行企业级加密时提供服务端私钥。 | 请参考备注 2。 |
| 0x10 (b'001111') | Wi-Fi 连接状态报告 | 通知手机 ESP 设备的 Wi-Fi 状态，包括 STA 状态和 SoftAP 状态。用于 STA 设备连接手机或 SoftAP。但是，当手机接收 Wi-Fi 状态时，除了本帧之外，还可以回复其他帧。 | data[0] 表示 opmode，包括：
  - 0x00: NULL
  - 0x01: STA
  - 0x02: SoftAP
  - 0x03: SoftAP & STA |

**备注：**

- 0x11: STA 设备的连接状态
- 0x21: STA 设备连接状态，0x0 表示处于连接状态并获取 IP 地址；0x1 表示处于连接状态但未获取 IP 地址；0x2 表示处于正在连接状态；0x3 表示处于连接状态但未获取 IP 地址。
备注：
- 备注 1: 数据的长度取决于数据长度字段。若传输方向是从 ESP 设备到手机，则表示向手机端提供所需信息。
- 备注 2: 数据的长度取决于数据长度字段。如果数据长度不够，该帧可用分片。

2. Frame Control
帧控制字段，占 1 字节，每个位表示不同含义。

<table>
<thead>
<tr>
<th>位</th>
<th>含义</th>
</tr>
</thead>
</table>
| 0x01 | 表示帧是否加密。
- 1 表示加密。
- 0 表示未加密。
该帧的加解密部分包括数据字段加密之前的完整明文数据（不包括校验部分）。控制帧暂不加密，故控制帧此位为 0。 |
| 0x02 | 该数据字段表示帧尾是否包含校验位，如 SHA1、MD5、CRC 等。该数据字段包含序列、数据长度以及明文。控制帧和数据帧都可以选择包含或不包含校验位。 |
| 0x04 | 表示数据方向。
- 0 表示传输方向是从手机到 ESP 设备。
- 1 表示传输方向是从 ESP 设备到手机。 |
| 0x08 | 表示是否要求对方回复 ACK。
- 0 表示不要求回复 ACK。
- 1 表示要求回复 ACK。 |
| 0x10 | 表示是否有后续的数据分片。
- 0 表示此帧没有后续数据分片。
- 1 表示还有后续数据分片，用来传输较长的数据。
对于分片帧，在数据字段的前两个字节中，会给出当前数据字节和随后内容部分的总长度（即最大支持 64K 的数据内容）。 |
| 0x10-0x80 | 保留 |

3. 序列控制
序列控制字段。帧发送时，无论帧的类型是什么，序列都会自动加 1，用来防止重放攻击 (Replay Attack)。每次重新连接后，序列清零。

4. 长度
数据字段的长度，不包含校验部分。

5. 数据
对于不同的类型或子类型，数据字段的含义均不同。请参考上方表格。

6. 校验
此字段占两个字节，用来校验序列、数据长度以及明文。

### 4.3.5 ESP32 端的安全实现

1. 数据安全
为了保证 Wi-Fi SSID 和密码的传输过程是安全的，需要使用对称加密算法 (例如 AES、DES 等) 对报文进行加密。在使用对称加密算法之前，需要使用非对称加密算法 (DH、RSA、ECC 等) 协商出 (或生成出) 一个共享密钥。

2. 保证数据完整性
为了保证数据完整性，需要加入校验算法，例如 SHA1、MD5、CRC 等。

3. 身份安全（签名）
某些算法如 RSA 可以保证身份安全。但如 DH 这类的算法，本身不能保证身份安全，需要添加其他算法来签名。

4. 防止重放攻击 (Replay Attack)
添加其到序列字段中，并且在数据校验过程中使用。
在 ESP32 端的代码中，你可以决定和开发加密密钥协商等安全处理的流程。手机应用向 ESP32 发送协商数据，数据会传送给应用层处理，如果应用层不处理，可使用 BluFi 提供的 DH 加密算法来协商密钥。

应用层需向 BluFi 注册以下几个与安全相关的函数:

```c
typedef void (*esp_blufi_negotiate_data_handler_t)(uint8_t *data, int len, uint8_t *output_data, int *output_len, bool *need_free)
```

该函数用来接收协商期间的正常数据(normal data)，数据处理完成后，需要将待发送的数据使用 output_data 和 output_len 传出。

BluFi 会在调用前 Negotiate_data_handler 后，发送 Negotiate_data_handler 传回的 output_data。

这里的两个 “*” 是因为需要发出去的数据长度未知，所以需要函数自行分配 (malloc) 或者指向全局变量，并告知是否需要通过 NEED_FREE 释放内存。

```c
typedef int (*esp_blufi_encrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int *crypt_len)
```

加密和解密的数据长度必须一致。其中 IV8 为帧的 8 位序列，可作为 IV 的某 8 个位来使用。

```c
typedef int (*esp_blufi_decrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int *crypt_len)
```

加密和解密的数据长度必须一致。其中 IV8 为帧的 8 位序列，可作为 IV 的某 8 个位来使用。

```c
typedef uint16_t (*esp_blufi_checksum_func_t)(uint8_t iv8, uint8_t *data, int len)
```

该函数用来进行校验，返回值为校验的值。BluFi 会使用该函数返回值与帧的校验值进行比较。

### 4.3.6 GATT 相关说明

**UUID**

BluFi Service UUID：0xFFFF，16 bit

BluFi（手机 -> ESP32）特性：0xFF01，主要权限：可写

BluFi（ESP32 -> 手机）特性：0xFF02，主要权限：可读可通知

### 4.4 引导加载程序 (Bootloader)

ESP-IDF 软件引导加载程序 (Bootloader) 主要执行以下任务:

1. 内部模块的最简化初始配置
2. 如果配置了 Flash 加密和/或 Secure，则对其进行初始化
3. 根据分区表和 ota_data (如果存在) 选择需要引导的应用程序 (app) 分区
4. 将此应用程序映象加载到 RAM (IRAM 和 DRAM) 中，最后把控制权转交给此应用程序

引导加载程序位于 flash 的 0x1000 偏移地址处。

关于启动过程以及 ESP-IDF 引导加载程序的更多信息，请参考 [应用程序的启动流程](#)

### 4.4.1 引导加载程序兼容性

建议使用最新发布的 ESP-IDF 版本。OTA（空中升级）更新可以在现场烧录新的应用程序,但不能烧录一个新的引导加载程序。因此，引导加载程序支持引导从 ESP-IDF 新版本中构建的应用程序。
但不支持引导从 ESP-IDF 旧版本中构建的程序。如果现有产品可能需要将应用程序降级到旧版本，那么在手动更新 ESP-IDF 时，请继续使用旧版本 ESP-IDF 引导加载程序的二进制文件。

备注：如果在生产中测试现有产品的 OTA 更新，请确保测试中使用的 ESP-IDF 引导加载程序二进制文件与生产中部署的相同。

ESP-IDF V2.1 之前的版本

与新版本相比，ESP-IDF V2.1 之前的版本构建的引导加载程序对硬件的配置更少。使用这些早期 ESP-IDF 版本的引导加载程序并构建新应用程序时，请启用配置选项 `CONFIG_APP_COMPATIBLE_PRE_V2_I_BOOTLOADERS`。

ESP-IDF V3.1 之前的版本

ESP-IDF V3.1 之前的版本构建的引导加载程序不支持分区表二进制文件中的 MD5 校验。使用这些 ESP-IDF 版本的引导加载程序并构建新应用程序时，请启用配置选项 `CONFIG_APP_COMPATIBLE_PRE_V3_I_BOOTLOADERS`。

配置 SPI Flash

每个 ESP-IDF 应用程序或引导加载程序的二进制文件中都包含一个文件头，其中内置了 `CONFIG_ESPTOOLPY_FLASHMODE`、`CONFIG_ESPTOOLPY_FLASHFREQ` 和 `CONFIG_ESPTOOLPY_FLASHSIZE`。这些是用于在启动时配置 SPI flash。ROM 中的一级引导程序从 flash 中读取二级引导程序文件头中配置信息，并使用这些信息来加载剩余的二级引导程序。然而，此时系统的时钟速度低于其被配置的速度，并且在这个阶段，只支持部分 flash 模式。因此，当二级引导程序运行时，它会从当前应用程序的二进制文件头中读取数据（而不是从引导加载程序的文件头中读取数据），并使用这些数据重新配置 flash。这样的配置流程可让 OTA 更新去更改当前使用的 SPI flash 的配置。

ESP-IDF V4.0 版本之前的引导加载程序使用其自身的文件头来配置 SPI flash，这意味着无法在 OTA 更新时更改 SPI flash 配置。为了与旧引导加载程序兼容，应用程序在其启动期间使用应用程序文件头中的配置信息重新初始化 flash 配置。

4.4.2 日志级别

引导加载程序日志的级别默认为“Info”。通过设置 `CONFIG_BOOTLOADER_LOG_LEVEL` 选项，可以增加或减少这个等级。这个日志级别与应用程序中使用的日志级别是分开的（见 Logging library）。

降低引导加载程序日志的详细程度可以稍微缩短整个项目的启动时间。

4.4.3 恢复出厂设置

在更新出现问题时，最好能有一种方法让设备回到已知的正常状态。这时可选择恢复出厂设置。

要回到原始出厂设置并清除所有用户设置，请在引导加载程序中配置 `CONFIG_BOOTLOADER_FACTORY_RESET`。

以下两种方式可以使设备恢复出厂设置。

- 清除一个或多个数据分区。`CONFIG_BOOTLOADER_DATA_FACTORY_RESET` 选项允许用户选择哪些数据分区在恢复出厂设置时需要被擦除。
- 用户可以使用以逗号分隔的列表形式指定分区的名称。为了提高可读性，可以选择添加空格（如：nvs, phy_init, nvs_custom）。

请确保选项里指定的分区名称和分区表中的名称相同。此处不能指定“app”类型的分区。
从“工厂”应用分区启动。当启用 CONFIG_BOOTLOADER_OTA_DATA_ERASE 选项，恢复出厂设置后，设备将从默认的“工厂”应用分区启动（如果分区表中没有“工厂”应用分区，则从默认的OTA应用分区启动）。这个恢复过程是通过擦除OTA数据分区来完成的，OTA数据分区中保存了当前选择的OTA分区槽。“工厂”应用分区槽（如果存在）永远不会通过OTA更新，因此重置为从“工厂”应用分区启动则意味着让固件应用程序恢复正常状态。

这两个配置选项都可以独立启用。

此外，下列配置选项用于配置触发恢复出厂设置的条件：

- **CONFIG_BOOTLOADER_NUM_PIN_FACTORY_RESET**: 输入管脚 (GPIO) 的编号。该管脚用于触发恢复出厂设置。必须在重置时将此管脚拉低或拉高（可配置）才能触发出厂重置事件。
- **CONFIG_BOOTLOADER_HOLD_TIME_GPIO**: 管脚电平保持时间（默认为5秒）。设备重置后，管脚电平必须保持该设定的时间，才能执行恢复出厂设置或引导测试分区（如适用）。
- **CONFIG_BOOTLOADER_FACTORY_RESET_PIN_LEVEL**: 设置管脚电平高低。设备重置后，根据此设置将管脚拉高或拉低，才能触发出厂重置事件。如果管脚具有内部上拉，则上拉会在管脚采样前生效。有关管脚内部上拉的详细信息，请参考 ESP32 的技术规格书。

### 4.4.4 从测试固件启动

用户可以编写特殊固件用于生产环境中测试，并在需要的时候运行。此时需要在项目分区表中专门申请一块分区用于保存该测试固件，其类型为 app，子类型为 test（详情请参考分区表）。

实现该测试应用固件需要为测试应用创建一个完全独立的 ESP-IDF 项目（ESP-IDF 中的每个项目仅构建一个应用）。该测试应用可以独立于主项目进行开发和测试，然后在生成测试时作为一个预编译的 bin 文件集成到主项目的测试应用程序分区的地址。

为了使主项目的引导加载程序支持这个功能，请设置 CONFIG_BOOTLOADER_APP_TEST 并配置以下两个选项：

- **CONFIG_BOOTLOADER_NUM_PIN_APP_TEST** - 设置启动 TEST 分区的管脚编号。选中的管脚将被配置为启动了内部上拉的输入，要触发测试应用，必须在重置时将此管脚拉低。

  当管脚输入被释放（则被拉高）并将设备重新启动后，正常配置的应用程序将启动（工厂或任意 OTA 应用分区槽）。

- **CONFIG_BOOTLOADER_HOLD_TIME_GPIO** - 设置 GPIO 电平保持的时间（默认为5秒）。设备重置后，管脚在设定的时间内必须持续保持低电平，然后才会执行出厂重置或引导测试分区（如适用）。

### 4.4.5 回滚

回滚和反回滚功能也必须在引导程序中配置。

请参考 OTA API 参考文档 中的应用程序回滚 和防回滚 章节。

### 4.4.6 看门狗

默认情况下，硬件 RTC 看门狗定时器在引导加载程序运行时保持运行。如果9秒后没有应用程序成功启动，它将自动重置芯片。

- 可以通过设置 CONFIG_BOOTLOADER_WDT_TIME_MS 并重新编译引导加载程序来调整超时时间。
- 可以通过调整应用程序的行为使 RTC 看门狗在应用程序启动后保持启动。看门狗需要由应用程序显示地重置（即“喂狗”），以避免重置。为此，请设置 CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE 选项，根据需要修改应用程序，然后重新编译应用程序。
- 通过禁用 CONFIG_BOOTLOADER_WDT_ENABLE 设置并重新编译引导加载程序，可以在引导加载程序中禁用 RTC 看门狗，但并不建议这样做。
4.4.7 引导加载程序大小

当前需要启用额外的引导加载程序功能，包括 Flash 回密或安全启动，尤其是设置高级别的 CONFIG_BOOTLOADER_LOG_LEVEL 时，监控引导加载程序.bin 文件的大小变得非常重要。

当使用默认的 CONFIG_PARTITION_TABLE_OFFSET 值 0x8000 时，二进制文件最大可为 0x7000 (28672) 字节。

如果引导加载程序二进制文件过大，则引导加载程序会构建失败并显示“Bootloader binary size [...] is too large for partition table offset”的错误。如果此二进制文件已经被烧录，那么 ESP32 将无法启动 - 日志中将记录无效分区表或无效引导加载程序校验和的错误。

可以使用如下方法解决此问题:

- 将 bootloader 编译器优化 重新设置回默认值“Size”。
- 降低引导加载程序日志级别。将日志级别设置为 Warning, Error 或 None 都会显著减少最终二进制文件的大小（但也可能会让调试变得更加困难）。
- 将 CONFIG_PARTITION_TABLE_OFFSET 设置为高于 0x8000 的值，以便稍后将分区表放置在 flash 中，这样可以增加引导加载程序的可用空间。如果分区表的 CSV 文件包含明确的分区偏移量，则需要修改这些偏移量，从而保证没有分区的偏移量低于 CONFIG_PARTITION_TABLE_OFFSET + 0x1000。（这包括随 ESP-IDF 提供的默认分区 CSV 文件）

当启用 Secure Boot V2 时，由于引导加载程序最先加载到固定大小的缓冲区中进行验证，对二进制文件大小的绝对限制为 48KB（0xC000 bytes）（不包括 4 KB 签名）。

4.4.8 从深度睡眠中快速启动

引导加载程序有 CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP 选项，可以减少从深度睡眠中唤醒的时间（有利于降低功耗）。当 CONFIG_SECURE_BOOT 选项禁用时，该选项可用。由于无需镜像校验，唤醒时间减少。在第一次启动时，引导加载程序将启动的应用程序的地址存储在 RTC FAST 存储器中。而在唤醒过程中，这个地址用于启动而无需任何检查，从而实现了快速加载。

4.4.9 自定义引导加载程序

用户可以扩展或修改当前的引导加载程序，具体有两种方法：使用钩子实现或重写覆盖当前程序。这两种方法在 ESP-IDF 示例的 custom_bootloader 文件夹中都有呈现。

- bootloader_hooks 介绍了如何将钩子与引导加载程序初始化连接。
- bootloader_override 介绍了如何覆盖引导加载程序的实现。

在引导加载程序的代码中，用户不能使用其他组件提供的驱动和函数，如果确实需要，请将该功能的实现部分放在项目中的 bootloader_components 目录中（注意，这会增加引导加载程序的大小）。

如果引导加载程序过大，则可能与内存中的分区表重叠，分区表默认烧录在偏移量 0x8000 处。增加分区表偏移量，将分区表放在 flash 中靠后的区域，这样可以增加引导程序的可用空间。

4.5 构建系统

本文档主要介绍 ESP-IDF 构建系统的实现原理以及组件等相关概念。如您想了解如何组织和构建新的 ESP-IDF 项目或组件，请阅读本文档。

4.5.1 概述

一个 ESP-IDF 项目可以看作是多个不同组件的集合，例如一个显示当前湿度的网页服务器会包含以下组件：

- ESP-IDF 基础库，包括 libc、ROM bindings 等
• Wi-Fi 驱动
• TCP/IP 协议栈
• FreeRTOS 操作系统
• 网页服务器
• 湿度传感器的驱动
• 负责将上述组件整合到一起的主程序

ESP-IDF 可以显式地指定和配置每个组件。在构建项目的时候，构建系统会前往 ESP-IDF 目录，项目目录和用户自定义组件目录（可选）中查找所有组件，允许用户通过文本菜单系统配置 ESP-IDF 项目中用到的每个组件。在所有组件配置结束后，构建系统开始编译整个项目。

概念

• 项目特指一个目录，其中包含了构建可执行应用程序所需的全部文件和配置，以及其他支持型文件，例如分区表、数据/文件系统分区和引导程序。
• 项目配置保存在项目根目录下名为 sdkconfig 的文件中，可以通过 idf.py menuconfig 进行修改，且一个项目只能包含一个项目配置。
• 应用程序由 ESP-IDF 构建得到的可执行文件。一个项目通常会构建两个应用程序：项目应用程序（可执行的主文件，即用户自定义的固件）和引导程序（启动并初始化项目应用程序）。
• 组件是模块化且独立的代码，会被编译成静态库 (.a 文件) 并链接到应用程序。部分组件由 ESP-IDF 官方提供，其他组件则来源于其它开源项目。
• 目标特指运行构建后应用程序的硬件设备。运行 idf.py --list-targets 可以查看当前 ESP-IDF 版本中支持目标的完整列表。

请注意，以下内容并不属于项目的组成部分：

• ESP-IDF 并不是项目的一部分，它独立于项目。通过 IDF_PATH 环境变量（保存 esp-idf 目录的路径）链接到项目，从而将 IDF 框架与项目分离。
• 交叉编译工具链并不是项目的组成部分。它应该被安装在系统 PATH 环境变量中。

4.5.2 使用构建系统

idf.py

idf.py 命令行工具提供了一个前端，可以帮助您轻松管理项目的构建过程。它管理了以下工具：

• CMake，配置待构建的项目
• Ninja，用于构建项目
• esptool.py，烧录目标硬件设备

可通过 idf.py 配置构建系统，具体可参考相关文档。

直接使用 CMake

为了方便，idf.py 已经封装了 CMake 命令，但是您愿意，也可以直接调用 CMake。

当 idf.py 在执行某些操作时，它会打印出其运行的每条命令以便参考。例如运行 idf.py build 命令与在 bash shell（或者 Windows Command Prompt）中运行以下命令是相同的：

```
mkdir -p build
cd build
cmake .. -G Ninja  # 或者 'Unix Makefiles'
ninja
```

在上面的命令列表中，cmake 命令对项目进行配置，并生成用于最终构建工具的构建文件。在这个例子中，最终构建工具是 Ninja: 运行 ninja 来构建项目。

没有必要多次运行 cmake。第一次构建后，往后每次只需运行 ninja 即可。如果项目需要重新配置，ninja 会自动重新调用 cmake。
若在 CMake 中使用 ninja 或 make，则多数 idf.py 子命令也会有其对应的目标，例如在构建目录下运行 make menuconfig 或 ninja menuconfig 与运行 idf.py menuconfig 是相同的。

备注：如果您已经熟悉了 CMake，那么可能会发现 ESP-IDF 的 CMake 构建系统不同寻常，为了减少样板文件，该系统封装了 CMake 的许多功能。请参考编写 CMake 组件以编写更多 “CMake 风格” 的组件。

使用 Ninja/Make 来烧录 您可以直接使用 ninja 或 make 运行如下命令来构建项目并烧录:

<table>
<thead>
<tr>
<th>命令</th>
</tr>
</thead>
<tbody>
<tr>
<td>ninja flash</td>
</tr>
<tr>
<td>或：</td>
</tr>
<tr>
<td>make app-flash</td>
</tr>
</tbody>
</table>

可用的目的还包括：flash、app-flash（仅用于 app)、bootloader-flash（仅用于 bootloader)。以这种方式烧录时，可以通过设置 ESPPORT 和 ESPBAUD 环境变量来指定串口设备和波特率。您可以在操作系统或 IDE 项目中设置该环境变量，或者直接在命令行中进行设置:

<table>
<thead>
<tr>
<th>命令</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPPORT=/dev/ttyUSB0 ninja flash</td>
</tr>
</tbody>
</table>

备注：在命令的开头为环境变量赋值属于 Bash shell 的语法，可在 Linux、macOS 和 Windows 的类 Bash shell 中运行，但在 Windows Command Prompt 中无法运行。

<table>
<thead>
<tr>
<th>命令</th>
</tr>
</thead>
<tbody>
<tr>
<td>或：</td>
</tr>
<tr>
<td>make -j3 app-flash ESPPORT=COM4 ESPBAUD=2000000</td>
</tr>
</tbody>
</table>

备注：在命令末尾为变量赋值属于 make 的语法，适用于所有平台的 make。

在 IDE 中使用 CMake

您还可以使用集成了 CMake 的 IDE，仅需将项目 CMakeLists.txt 文件的路径告诉 IDE 即可。集成 CMake 的 IDE 通常会有自己的构建工具 (CMake 称之为 “生成器”)，它是组成 IDE 的一部分，用来构建源文件。

向 IDE 中添加除 build 目标以外的自定义目标（如添加 “Flash” 目标到 IDE）时，建议调用 idf.py 命令来执行这些 “特殊” 的操作。

有关将 ESP-IDF 同 CMake 集成到 IDE 中的详细信息，请参阅构建系统的元数据。

设置 Python 解释器

ESP-IDF 适用于 Python 3.7 以上版本。

idf.py 和其他的 Python 脚本会使用默认的 Python 解释器运行。如 python。您可以通过 python3 $IDF_PATH/tools/idf.py ... 命令切换到别的 Python 解释器，或者您可以通过设置 shell 别名或其他脚本来简化该命令。

如果直接使用 CMake，运行 cmake -D PYTHON=python3 ...，CMake 会使用传入的值覆盖默认的 Python 解释器。

如果使用集成 CMake 的 IDE，可以在 IDE 的图形用户界面中给名为 PYTHON 的 CMake cache 变量设置新的值来覆盖默认的 Python 解释器。
如果想在命令行中更优雅地管理 Python 的各个版本，请查看 pyenv 或 virtualenv 工具，它们会帮助您更改默认的 python 版本。

4.5.3 示例项目

示例项目的目录树结构可能如下所示:

```
- myProject/
  - CMakeLists.txt
  - sdkconfig
  - components/
    - component1/
    - Kconfig
      - src1.c
    - component2/
      - Kconfig
      - src1.c
      - include/
    - main/
      - CMakeLists.txt
      - src1.c
      - src2.c
    - build/
```

该示例项目“myProject”包含以下组成部分:

- 顶层项目 CMakeLists.txt 文件，这是 CMake 用于学习如何构建项目的主要文件，可以在这个文件中设置项目全局的 CMake 参数。顶层项目 CMakeLists.txt 文件会导入/tools/cmake/project.cmake 文件，由它负责实现构建系统的其余部分。该文件最后会设置项目的名称，并定义该项目。
- “sdkconfig”配置文件，执行 idf.py menuconfig 时会创建或更新此文件，文件中保存了项目中所有组件（包括 ESP-IDF 本身）的配置信息。sdkconfig 文件可能会也可能不会被添加到项目的源码管理系统中。
- 可选的“components”目录中包含了项目的各个自定义组件，并不是每个项目都需要这种自定义组件，但它有助于构建可复用的代码或者构建第三方（不属于 ESP-IDF）的组件。或者，您也可以在顶层 CMakeLists.txt 中设置 EXTRA_COMPONENT_DIRS 变量以查找其他自定义组件。
- “main”目录是一个特殊的组件，它包含项目本身的源代码。main 是默认名称。CMake 变量 COMPONENT_DIRS 默认包含此组件，但您可以修改此变量。有关详细信息，请参阅命名 main 组件。如果项目中源文件较多，建议将其归于组件中，而不是全部放在“main”中。
- “build”目录是存放构建输出的地方，如果没有此目录，idf.py 会自动创建。CMake 会配置项目，并在此目录下生成临时的构建文件。随后，在主构建过程的运行期间，该目录还会保存临时目标文件、库文件以及最后输出的二进制文件。此目录通常不会添加到项目的源码管理系统中，也不会随项目源码一同发布。

每个组件目录都包含一个 CMakeLists.txt 文件，里面会定义一些变量以控制该组件的构建过程，以及其与整个项目的集成。更多详细信息请参阅组件 CMakeLists 文件。

每个组件还可以包含一个 Kconfig 文件，它用于定义 menuconfig 时显示的组件配置选项。某些组件可能还会包含 Kconfig.projbuild 和 project_include.cmake 特殊文件，它们用于覆盖项目的其他部分。

4.5.4 项目 CMakeLists 文件

每个项目都有一个顶层 CMakeLists.txt 文件，包含整个项目的构建设置。默认情况下，项目 CMakeLists 文件会非常小。

最小 CMakeLists 文件示例

最小项目:
必要部分

每个项目都要按照上面显示的顺序添加上述三行代码:

- `cmake_minimum_required(VERSION 3.16)` 必须放在 CMakeLists.txt 文件的第一行，它会告诉 CMake 构建该项目所需的最小版本号。ESP-IDF 支持 CMake 3.16 或更高的版本。
- `include($ENV{IDF_PATH}/tools/cmake/project.cmake)` 会导入 CMake 的其余功能来完成配置项目、检测组件等任务。
- `project(myProject)` 会创建项目本身，并指定项目名称。该名称会作为最终输出的二进制文件的名字，即 myProject.elf 和 myProject.bin。每个 CMakeLists 文件只能定义一个项目。

可选的项目变量

以下这些变量都有默认值，用户可以覆盖这些变量值以自定义构建行为。更多实现细节，请参阅 /tools/cmake/project.cmake 文件。

- `COMPONENT_DIRS`：组件的搜索目录，默认为 IDF_PATH/components、PROJECT_DIR/components、和 `EXTRA_COMPONENT_DIRS`。如果您不想在这些位置搜索组件，请覆盖此变量。
- `EXTRA_COMPONENT_DIRS`：用于搜索组件的其它可选目录列表。路径可以是相对于项目目录的相对路径，也可以是绝对路径。
- `COMPONENTS`：要构建项目中的组件名称列表，默认为 `COMPONENT_DIRS` 目录下搜索到的所有组件。使用此变量可以“精简”项目以缩短构建时间。请注意，如果一个组件通过 `COMPONENT_REQUIRES` 指定了它依赖的另一个组件，则会自动将其添加到 `COMPONENTS` 中，所以 `COMPONENTS` 列表可能会非常短。

以上变量中的路径可以是绝对路径，或者是相对于项目目录的相对路径。

请使用 cmake 中的 `set` 命令来设置这些变量，如 `set(VARIABLE "VALUE")`。请注意，`set()` 命令需放在 `include(...)` 之前，`cmake_minimum(...)` 之后。

重命名 main 组件

构建系统会对 main 组件进行特殊处理。假如 main 组件位于预期的位置（即 $PROJECT_PATH/main），那么它会被自动添加到构建系统中。其他组件也会作为其依赖项被添加到构建系统中，这使用户免于处理依赖关系，并提供即时可用的构建功能。重命名 main 组件会略微上述这些步骤。组件名称、重命名 main组件的步骤如下:

1. 重命名 main 目录。
2. 在项目 CMakeLists.txt 文件中设置 `EXTRA_COMPONENT_DIRS`，并添加重命名后的 main 目录。
3. 在组件的 CMakeLists.txt 文件中设置 `COMPONENT_REQUIRES` 或 `COMPONENT_PRIV_REQUIRES` 以指定依赖项。

覆盖默认的构建规范

构建系统设置了一些全局的构建规范（编译标志、定义等），这些规范可用于编译来自所有组件的所有源文件。

例如，其中一个默认的构建规范是编译选项 Wextra。假设一个用户想用 Wno-extra 来覆盖这个选项，应在 `project()` 之后进行:
这确保了用户设置的编译选项不会被默认的构建规范所覆盖，因为默认的构建规范是在 project() 内设置的。

4.5.5 组件 CMakeLists 文件

每个项目都包含一个或多个组件，这些组件可以是 ESP-IDF 的一部分，可以是项目自身组件目录的一部分，也可以从自定义组件目录添加（见上文）。

组件是 COMPONENT_DIRS 列表中包含 CMakeLists.txt 文件的任何目录。

搜索组件

搜索 COMPONENT_DIRS 中的目录列表以查找项目的组件，此列表中的目录可以是组件自身（即包含 CMakeLists.txt 文件的目录），也可以是子目录为组件的顶级目录。

当 CMake 运行项目配置时，它会记录本次构建包含的组件列表，它可用于调试某些组件的添加/排除。

同名组件

ESP-IDF 在搜索所有待构建的组件时，会按照 COMPONENT_DIRS 指定的顺序依次进行，这意味着在默认情况下，首先搜索 ESP-IDF 内部组件（IDF_PATH/components），然后是 EXTRA_COMPONENT_DIRS 中的组件。最后是项目组件（PROJECT_DIR/components）。如果这些目录中的两个或者多个包含具有相同名字的组件，则使用搜索到最后一个位置的组件。这就允许将组件复制到项目目录中再修改以覆盖 ESP-IDF 组件，如果使用这种方式，ESP-IDF 目录本身可以保持不变。

备注：如果在现有项目中通过将组件移动到一个新位置来覆盖它，项目不会自动看到新组件的路径。请运行 idf.py reconfigure 命令后（或删除项目构建文件夹）再重新构建。

最小组件 CMakeLists 文件

最小组件 CMakeLists 文件通过使用 idf_component_register 将组件添加到构建系统中。

```cmake
idf_component_register(SRCS "foo.c" "bar.c" INCLUDE_DIRS "include" REQUIRES mbedtls)
```

- SRCS 是源文件列表（*.c、*.cpp、*.cc、*.S），里面所有的源文件都将编译进组件库中。
- INCLUDE_DIRS 是目录列表，里面的路径会被添加到所有需要该组件的组件（包括 main 组件）全局 include 搜索路径中。
- REQUIRES 实际上并不是必需的，但通常需要它来声明该组件需要使用哪些其它组件，请参考组件依赖。

上述命令会构建生成与组件同名的库，并最终被链接到应用程序中。

上述目录通常设置为相对于 CMakeLists.txt 文件的相对路径。当然也可以设置为绝对路径。

还有其它参数可以传递给 idf_component_register，具体可参考 here。

有关更完整的 CMakeLists.txt 示例，请参阅组件依赖示例 和组件 CMakeLists 示例。
预设的组件变量

以下专用于组件的变量可以在组件 CMakeLists 中使用，但不建议修改：

- COMPONENT_DIR: 组件目录，即包含 CMakeLists.txt 文件的绝对路径，它与
  CMAKE_CURRENT_SOURCE_DIR 变量一致，路径中不能包含空格。
- COMPONENT_NAME: 组件名，与组件目录名相同。
- COMPONENT_ALIAS: 库别名，由构建系统在内部为组件创建。
- COMPONENT_LIB: 库名，由构建系统在内部为组件创建。

以下变量在配置级别中被设置，但可在组件 CMakeLists 中使用：

- CONFIG_*: 项目配置中的每个值在 cmake 中对应一个以 CONFIG_ 开头的变量，更多详细信息
  请参阅 Kconfig。
- ESP_PLATFORM: ESP-IDF 构建系统处理 CMake 文件时，其值设为 1。

构建/项目变量

以下是可作为构建属性的构建/项目变量，可通过组件 CMakeLists.txt 中的 idf_build_get_property
查询其变量值。

- PROJECT_NAME: 项目名，在项目 CMakeLists.txt 文件中设置。
- PROJECT_DIR: 项目目录（包含项目 CMakeLists 文件）的绝对路径，与
  CMAKE_SOURCE_DIR 变量相同。
- COMPONENTS: 此次构建中包含的所有组件的名称，具体格式为用分号隔开的 CMake 列表。
- IDF_VER: ESP-IDF 的 git 版本号，由 git describe 命令生成。
- IDF_VERSION_MAJOR、IDF_VERSION_MINOR、IDF_VERSION_PATCH: ESP-IDF 的组件版本，
  可用于条件表达式。请注意这些信息的精确度不如 IDF_VER 变量，版本号 v4.0-dev-*, v4.0-
  beta1, v4.0-rc1 和 v4.0 对应的 IDF_VERSION_* 变量值是相同的，但是 IDF_VER 的值是
  不同的。
- IDF_TARGET: 项目的硬件目标名称。
- PROJECT_VER: 项目版本号。
  - 如果设置 CONFIG_APP_PROJECT_VER_FROM_CONFIG 选项，将会使用
    CONFIG_APP_PROJECT_VER 的值。
  - 或者，如果在项目 CMakeLists.txt 文件中设置了 PROJECT_VER 变量，则该变量值可以使用。
  - 或者，如果 PROJECT_DIR/version.txt 文件存在，其内容会用作 PROJECT_VER 的值。
  - 或者，如果项目位于某个 Git 仓库中，则使用 git describe 命令的输出作为 PROJECT_VER
    的值。
  - 否则 PROJECT_VER 的值为 1。
- EXTRA_PARTITION_SUBTYPES: CMake 列表，用于创建额外的分区子类型。子类型的描述由字
  符串组成，以逗号为分隔，格式为 type_name, subtype_name, numeric_value。组件可通过
  此列表，添加新的子类型。
其它与构建属性有关的信息请参考这里。

组件编译控制

在编译特定组件的源文件时，可以使用 target_compile_options 函数来传递编译器选项:

```cmake
target_compile_options(${COMPONENT_LIB} PRIVATE -Wno-unused-variable)
```

如果给单个源文件指定编译器标志，可以使用 CMake 的 set_source_files_properties 命令:

```cmake
set_source_files_properties(mysrc.c
  PROPERTIES COMPILE_FLAGS
  -Wno-unused-variable)
```

如果上游代码在编译的时候发出了警告，那么这样做可能会很有效。

请注意，上述两条命令只能在组件 CMakeLists 文件的 idf_component_register 命令之后调用。

Espressif Systems 2135 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
4.5.6 组件配置

每个组件都可以包含一个 Kconfig 文件，和 CMakeLists.txt 放在同一目录下。Kconfig 文件中包含要添加到该组件配置菜单中的一些配置设置信息。

运行 menuconfig 时，可以在 Component Settings 菜单栏下找到这些设置。

创建一个组件的 Kconfig 文件，最简单的方法就是使用 ESP-IDF 中现有的 Kconfig 文件作为模板，在这基础上进行修改。

有关示例请参阅添加条件配置。

4.5.7 预处理器定义

ESP-IDF 构建系统会在命令行中添加以下 C 预处理器定义:

- ESP_PLATFORM：可以用来检测在 ESP-IDF 内发生了构建行为。
- IDF_VER：定义 git 版本字符串，例如: v2.0 用于标记已发布的版本，v1.0-275-g0efaa4f 则用于标记任意某次的提交记录。

4.5.8 组件依赖

编译各个组件时，ESP-IDF 系统会递归评估其依赖项。这意味着每个组件都需要声明它所依赖的组件，即“requires”。

编译组件

```c
idf_component_register(...
    REQUIREES mbedtls
    PRIV_REQUIRES console spiffs)
```

- REQUIREES 需要包含所有在当前组件的公共头文件里 #include 的头文件所在的组件。
- PRIV_REQUIRES 需要包含被当前组件的源文件 #include 的头文件所在的组件 (除非已经被设置在了 REQUIREES 中)。以及是当前组件正常工作必须要链接的组件。
- REQUIREES 和 PRIV_REQUIRES 的值不能依赖于任何配置选项(CONFIG_xxx 宏)，这是因为在配置加载之前，依赖关系就已经被展开。其它组件变量 (比如包含路径或源文件) 可以依赖配置选择。
- 如果当前组件除了通用组件依赖项 中设置的通用组件（比如 RTOS, libc 等）外，并不依赖其它组件，那么对于上述两个 REQUIREES 变量，可以选择其中一个或是两个都不设置。

如果组件仅支持某些硬件目标（IDF_TARGET 的值），则可以在 idf_component_register 中指定 REQUIRED_IDF_TARGETS 来声明这个需求。在这种情况下，如果构建系统导入了不支持当前硬件目标的组件时就会报错。

备注：在 CMake 中，REQUIREES 和 PRIV_REQUIRES 是 CMake 函数 target_link_libraries(... PUBLIC ...) 和 target_link_libraries(... PRIVATE ...) 的近似包装。

组件依赖示例

假设现在有一个 car 组件，它需要使用 engine 组件，而 engine 组件需要使用 spark_plug 组件:

```bash
- autoProject/
  - CMakeLists.txt
  - components/ - car/ - CMakeLists.txt
    - car.c
    - car.h
```

(下页继续)
**Car组件** car.h 头文件是 car 组件的公共接口。该头文件直接包含了 engine.h, 这是因为它需要使用 engine.h 中的一些声明:

```c
/* car.h */
#include "engine.h"

#ifndef ENGINE_IS_HYBRID
#define CAR_MODEL "Hybrid"
#endif
```

同时 car.c 也包含了 car.h:

```c
/* car.c */
#include "car.h"
```

这代表文件 car/CMakeLists.txt 需要声明 car 需要 engine:

```c
idf_component_register(SRCS "car.c"
  INCLUDE_DIRS "."
  REQUIRES engine)
```

- SRCS 提供 car 组件中源文件列表。
- INCLUDE_DIRS 提供该组件公共头文件目录列表，由于 car.h 是公共接口，所以这里列出了所有包含了 car.h 的目录。
- REQUIRES 给出该组件的公共接口所需的组件列表。由于 car.h 是一个公共头文件并且包含了来自 engine 的头文件，所以我们这里包含 engine。这样可以确保任何包含 car.h 的其他组件也能递归地包含所需的 engine.h。

**Engine组件** engine 组件也有一个公共头文件 include/engine.h，但这个头文件更为简单:

```c
/* engine.h */
#define ENGINE_IS_HYBRID

void engine_start (void);
```

在 engine.c 中执行:

```c
/* engine.c */
#include "engine.h"
#include "spark_plug.h"
...
```

在该组件中，engine 依赖于 spark_plug, 但这是私有依赖关系。编译 engine.c 需要 spark_plug.h 但不需要包含 engine.h。

这代表文件 engine/CMakeLists.txt 可以使用 PRIV_REQUIRES:

```c
idf_component_register(SRCS "engine.c"
  INCLUDE_DIRS "include"
  PRIV_REQUIRES spark_plug)
```
因此，car 组件中的源文件不需要在编译器搜索路径中添加 spark_plug include 目录。这可以加快编译速度，避免编译器命令行过多的冗长。

**Spark Plug 组件**
spark_plug 组件没有依赖项，它有一个公共头文件 spark_plug.h，但不包含其他组件的头文件。

这代表 spark_plug/CMakeLists.txt 文件不需要任何 REQUIRES 或 PRIV_REQUIRES:

```
idf_component_register(SRCS "spark_plug.c"
            INCLUDE_DIRS ".")
```

**源文件 Include 目录**
每个组件的源文件都是用这些 Include 路径目录编译的，这些路径在传递给 idf_component_register 的参数中指定:

```
idf_component_register(. .
            INCLUDE_DIRS "include"
            PRIV_INCLUDE_DIRS "other")
```

- 当前组件的 INCLUDE_DIRS 和 PRIV_INCLUDE_DIRS。
- REQUIRES 和 PRIV_REQUIRES 参数指定的所有其他组件（即当前组件的所有公共和私有依赖项）所设置的 INCLUDE_DIRS。
- 递归列出所有组件 REQUIRES 列表中 INCLUDE_DIRS 目录（如递归展开这个组件的所有公共依赖项）。

**主要组件依赖项**

main 组件比较特别，因为它在构建过程中自动依赖所有其他组件。所以不需要向这个组件传递 REQUIRE 或 PRIV_REQUIRE。有关不再使用 main 组件时需要更改哪些内容，请参考重命名 main 组件。

**通用组件依赖项**

为避免重复性工作，各组件都用自动依赖一些“通用”IDF 组件，即使它们没有被明确提及。这些组件的头文件会一直包含在构建系统中。

通用组件包括: cxx, newlib, freertos, esp_hw_support, heap, log, soc, hal, esp_rom, esp_common, esp_system。

**在构建中导入组件**

- 默认情况下，每个组件都会包含在构建系统中。
- 如果将 COMPONENTS 变量设置为项目直接使用的最小组件列表，那么构建系统会扩展到包含所有组件。完整的组件列表为:
  - COMPONENTS 中明确提及的组件。
  - 这些组件的依赖项（以及递归运算后的组件）。
  - 每个组件都依赖的通用组件。
- 将 COMPONENTS 设置为所需组件的最小列表，可以显著减少项目的构建时间。

**循环依赖**

一个项目中可能包含组件 A 和组件 B，而组件 A 依赖（REQUIRES 或 PRIV_REQUIRES）组件 B，组件 B 又依赖组件 A。这就是所谓的依赖循环或循环依赖。
Chapter 4. API

CMake 通常会在链接器命行上重复两次组件库名称来自动处理循环依赖。然而这种方法并不总是有效，还是可能构建失败并出现关于“Undefined reference to ...”的链接器错误。这通常是由引用了循环依赖中某一组件中定义的符号，或者存在较大的循环依赖关系，即 A -> B -> C -> D -> A。这种情况极有可能发生。

最好的解决办法是重构组件以消除循环依赖关系。在大多数情况下，没有循环依赖的软件架构具有模块化和分层清晰的特性，并且从长远来看更容易维护。然而，移除循环依赖关系并不容易做到。

要绕过由循环依赖引起的链接器错误，最简单的解决方法是增加其中一个组件库的 CMake

\texttt{LINK\_INTERFACE\_MULTIPICITY} 属性。这会让 CMake 在链接器命行上对此库及其依赖项重复

两次以上。

例如:

\begin{verbatim}
set_property(TARGET $\{COMPONENT\_LIB\} APPEND_PROPERTY LINK\_INTERFACE\_MULTIPICITY 3)
\end{verbatim}

- 这一行应该放在组件 CMakeLists.txt 文件 idf\_component\_register 之后。
- 可以的话，将此行放置在依稀其他组件而造成循环依赖的组件中。实际上，该行可以在循环内的任何一个组件中，但建议将其放置在拥有链接器错误提示信息中显示的源文件的组件中，或是放置在定义了链接器错误提示信息中所提及的符号的组件，先从这些组件开始是个不错的选择。
- 通常将值增加到 3（默认值是 2）就足够了，但如果不起作用，可以尝试逐步增加这个数字。
- 注意，增加这个选项会使链接器的命令行变长，链接阶段变慢。

高级解决方法：未定义符号 如果只有一两个符号导致循环依赖，而所有其他依赖都是线性的，那么有一种替代方法可以避免链接器错误：在链接时将“反向”依赖所需的特定符号指定为未定义符号。

例如，如果组件 A 依赖于组件 B，但组件 B 也需要引用组件 A 的 reverse\_ops。可以这样来添加如下一行，以在链接时避免这出现循环。

\begin{verbatim}
# 该符号是由“组件 A”在链接时提供
target_link_libraries($\{COMPONENT\_LIB\} INTERFACE "-u reverse_ops")
\end{verbatim}

- u 参数意味着链接器将始终在链接中包含此符号，而不管依赖项顺序如何。
- 该行应该放在组件 CMakeLists.txt 文件中的 idf\_component\_register 之后。
- 如果“组件 B”不需要访问“组件 A”的任何头文件，只需链接几个符号，那么这一行可以用来代替 B 对 A 的任何 REQUIRE。

请参考 target\_link\_libraries 文档以了解更多关于此 CMake 函数的信息。

构建系统中依赖处理的实现细节

- 在 CMake 配置进程的早期阶段会运行 expand\_requirements.cmake 脚本。该脚本会对所有组件的 CMakeLists.txt 文件进行局部的运算，得到一张组件依赖关系图（此图可能会有闭环）。此图用于在构建目录中生成 component\_depends.cmake 文件。
- CMake 主进程会导入该文件，并以此来确定要包含到构建系统中的组件列表（内部使用的 BUILD\_COMPONENTS 变量）。BUILD\_COMPONENTS 变量由排好序，依赖组件排在前面，由于组件依赖关系图中可能存在闭环，因此不能保证每个组件都满足该排序规则。如果给定相同的组件集和依赖关系，那么最终的排序结果应该是确定的。
- CMake 会将 BUILD\_COMPONENTS 的值以“Component names:”的形式打印出来。
- 然后执行构建系统中包含的每个组件的配置。
- 每个组件都被正常包含在构建系统中，然后再次执行 CMakeLists.txt 文件，将组件库加入构建系统。

组件依赖顺序 BUILD\_COMPONENTS 变量中组件的顺序决定了构建过程中的其它顺序，包括：

- 项目导入 project\_include\_cmake 文件的顺序。
- 生成用于编译（通过 -I 参数）的头文件路径列表的顺序。请注意，对于给定组件的源文件，仅需将该组件的依赖组件的头文件路径告知编译器。
4.5.9 覆盖项目的部分设置

project_include.cmake

如果组件的某些构建行为需要在组件 CMakeLists 文件之前被执行，您可以在组件目录下创建名为
project_include.cmake 的文件，project.cmake 在运行过程中会导入此 CMake 文件。

project_include.cmake 文件在 ESP-IDF 内部使用，以定义项目范围内的构建功能。比如 esptool.
py 的命令行参数和 bootloader 这个特殊的应用程序。

与组件 CMakeLists.txt 文件有所不同，在导入“project_include.cmake”文件的时候，当前源文件目录
（即 CMAKE_CURRENT_SOURCE_DIR `和工作目录）为项目目录。如果想获得当前组件的绝对路径，可
以使用 `\ COMPONENT_PATH 变量。

请注意，project_include.cmake 对于大多数常规的组件并不是必需的。例如给项目添加 include 搜
索目录，给最终的链接步骤添加 LDFLAGS 选项等等都可以通过 CMakeLists.txt 文件来自定义。详细
信息请参考可选的项目变量。

project_include.cmake 文件会按照 BUILD_COMPONENTS 变量中组件的顺序（由 CMake 记录）依
次导入，即只有在当前组件所有依赖组件的 project_include.cmake 文件都被导入后，当前组件
的 project_include.cmake 文件才会被导入，除非两个组件在同一个依赖闭环中。如果某个
project_include.cmake 文件依赖于另一组件设置的变量，则要特别注意上述情况。更多详情请参
阅构建系统中依赖处理的实现细节。

在 project_include.cmake 文件中设置变量或目标时要格外小心，这些值被包含在项目的顶层
CMake 文件中，因此他们会影响或破坏所有组件的功能。

KConfig.projbuild

与 project_include.cmake 类似，也可以为组件定义一个 KConfig 文件以实现全局的组件配置。如
果要在 menuconfig 的顶层添加配置选项，而不是在“Component Configuration”子菜单中，则可以在
CMakeLists.txt 文件所在目录的 KConfig.projbuild 文件中定义这些选项。

在此文件中添加配置时要小心，因为这些配置会包含在整个项目配置中。在可能的情况下，请为 组件配
置 创建 KConfig 文件。

project_include.cmake 文件在 ESP-IDF 内部使用，以定义项目范围内的构建功能，比如 esptool.
py 的命令行参数和 bootloader 这个特殊的应用程序。

通过封装现有函数进行重新定义或扩展

链接器具有封装功能，可以重新定义或扩展现有 ESP-IDF 函数的行为。如需封装函数，您需要在项目的
CMakeLists.txt 文件中提供以下 CMake 声明：

```
target_link_libraries($(COMPONENT_LIB) INTERFACE "-Wl,--wrap=function_to_redefine")
```

其中，function_to_redefine 为需要被重新定义或扩展的函数名称。启用此选项后，链接器将把二
进制库中所有对 function_to_redefine 函数的调用改为对 __wrap_function_to_redefine 函
数的调用。因此，您必须在应用程序中定义这一符号。

链接器会提供一个名为 __real_function_to_redefine 的新符号，指向将被重新定义的函数的原
有实现。由此，可以从新的实现中调用该函数，从而对原有实现进行扩展。

请参考 build_system/wrappers 示例，了解其详细原理。更多细节请参阅 examples/build_system/wrappers/README.md。

4.5.10 仅配置组件

仅配置组件是一类不包含源文件的特殊组件，仅包含 KConfig projbuild、
KConfig 和 CMakeLists.txt 文件，该 CMakeLists.txt 文件仅有一行代码，调用

CMake_CURRENT_SOURCE_DIR`
idf_component_register() 函数。此函数会将组件导入到项目构建中，但不会构建任何库，也不会将头文件添加到任何 include 搜索路径中。

### 4.5.11 CMake 调试

请查看 CMake v3.16 官方文档 获取更多关于 CMake 和 CMake 命令的信息。

调试 ESP-IDF CMake 构建系统的一些技巧：

- CMake 运行时，会打印大量诊断信息，包括组件列表和组件路径。
- 运行 `cmake -DDEBUG=1, IDF` 构建系统会生成更详细的诊断输出。
- 运行 `cmake` 时指定 `--trace` 或 `--trace-expand` 选项会提供大量有关控制流信息。详情请参考 CMake 命令行文档。

当从项目 CMakeLists 文件导入时，project.cmake 文件会定义工具模块和全局变量，并在系统环境中没有设置 `IDF_PATH` 时设置 `IDF_PATH`。

同时还定义了一个自定义版本的内置 CMake project 函数，这个函数被覆盖，以添加所有 ESP-IDF 特定的项目功能。

#### 警告未定义的变量

默认情况下，警告未定义的变量这一功能是关闭的。

可通过将 `--warn-uninitialized` 标志传递给 CMake 或通过将 `--cmake-warn-uninitialized` 传递给 idf.py 来使能这一功能。这样，如果在构建的过程中引用了未定义的变量，CMake 会打印警告。这对查找有错误的 CMake 文件非常有用。

更多信息，请参考文件 `/tools/cmake/project.cmake` 以及 `/tools/cmake/` 中支持的函数。

### 4.5.12 组件 CMakeLists 示例

因为构建环境试图设置大多数情况都能工作的合理默认值，所以组件 CMakeLists.txt 文件可能非常小，甚至是空的，请参考最小组件 CMakeLists 文件。但有些功能往往需要覆盖预设的组件变量 才能实现。

以下是组件 CMakeLists 文件的更高级的示例。

#### 添加条件配置

配置系统可用于根据项目配置中选择的选项有条件地编译某些文件。

Kconfig:

```
cfg FOO_ENABLE_BAR
    bool "Enable the BAR feature."
    help
        This enables the BAR feature of the FOO component.
```

CMakeLists.txt:

```
set(srcs "foo.c" "more_foo.c")

if(CONFIG_FOO_ENABLE_BAR)
    list(APPEND srcs "bar.c")
endif()

idf_component_register(SRCS "${srcs}"
    ...
```
上述示例使用了 CMake 的 if 函数和 list APPEND 函数。
也可用于选择或删除某一实现，如下所示:

Kconfig:

```cmake
config ENABLE_LCD_OUTPUT
  bool "Enable LCD output."
  help
    Select this if your board has a LCD.
config ENABLE_LCD_CONSOLE
  bool "Output console text to LCD"
  depends on ENABLE_LCD_OUTPUT
  help
    Select this to output debugging output to the lcd
config ENABLE_LCD_PLOT
  bool "Output temperature plots to LCD"
  depends on ENABLE_LCD_OUTPUT
  help
    Select this to output temperature plots
```

CMakeLists.txt:

```cmake
if(CONFIG_ENABLE_LCD_OUTPUT)
  set(srcs lcd-real.c lcd-spi.c)
else()
  set(srcs lcd-dummy.c)
endif()

# 如果启用了控制台输出功能，则需要加入字体
if(CONFIG_ENABLE_LCD_CONSOLE OR CONFIG_ENABLE_LCD_PLOT)
  list(APPEND srcs "font.c")
endif()

idf_component_register(SRCS "${srcs}"
  ...
)
```

硬件目标的条件判断

CMake 文件可以使用 IDF_TARGET 变量来获取当前的硬件目标。
此外，如果当前的硬件目标是 xyz（即 IDF_TARGET=xyz），那么 Kconfig 变量
CONFIG_IDF_TARGET_XYZ 同样也会被设置。
请注意，组件可以依赖 IDF_TARGET 变量，但不能依赖这个 Kconfig 变量。同样也不可在 CMake 文件的
include 语句中使用 Kconfig 变量。在这种上下文中可以使用 IDF_TARGET。

生成源代码

有些组件的源文件可能并不是由组件本身提供，而必须从另外的文件生成。假设组件需要一个头文件，该
文件由 BMP 文件转换后（使用 bmp2h 工具）的二进制数据组成，然后将头文件包含在名为 graphics_lib.c
的文件中:

```cmake
add_custom_command(OUTPUT logo.h
  COMMAND bmp2h -i ${COMPONENT_DIR}/logo.bmp -o log.h
  DEPENDS ${COMPONENT_DIR}/logo.bmp
  VERBATIM)
add_custom_target(logo DEPENDS logo.h)
```

(下页继续)
add_dependencies(${COMPONENT_LIB} logo)

set_property(DIRECTORY "${COMPONENT_DIR}" APPEND PROPERTY ADDITIONAL_MAKE_CLEAN_FILES logo.h)

这个示例改编自 CMake 的一则 FAQ，其中还包含了一些同样适用于 ESP-IDF 构建系统的示例。

这个示例会在当前目录（构建目录）中生成 logo.h 文件，而 logo.bmp 会随组件一起提供在组件目录中。因为 logo.h 是一个新生成的文件，一旦项目需要清理，该文件也应该被清除。因此，要将该文件添加到 ADDITIONAL_MAKE_CLEAN_FILES 属性中。

备注：如果需要生成文件作为项目 CMakeLists.txt 的一部分，而不是作为组件 CMakeLists.txt 的一部分，此时需要使用 ${PROJECT_PATH} 替代 ${COMPONENT_DIR}，使用 ${PROJECT_NAME}.elf 替代 ${COMPONENT_LIB}。

如果某个源文件是从其他组件中生成，且包含 logo.h 文件，则需要调用 add_dependencies，在这两个组件之间添加一个依赖项，以确保组件源文件按照正确顺序进行编译。

嵌入二进制数据

有时您的组件希望使用一个二进制文件或者文本文件，但是您又不想将它们重新格式化为 C 源文件。这时，您可以在组件注册中指定 EMBED_FILES 参数，用空格分隔要嵌入的文件名称：

idf_component_register(...
  EMBED_FILES server_root_cert.der)

或者，如果文件是字符串，则可以使用 EMBED_TXTFILES 变量，把文件的内容转成以 null 结尾的字符串嵌入：

idf_component_register(...
  EMBED_TXTFILES server_root_cert.pem)

文件的内容会被添加到 Flash 的 .rodata 段，用户可以通过符号名来访问。如下所示：

extern const uint8_t server_root_cert_pem_start[] asm("_binary_server_root_cert_pem_start");
extern const uint8_t server_root_cert_pem_end[] asm("_binary_server_root_cert_pem_end");

符号名会根据文件名全生成，如 EMBED_FILES 中所示，字符 /.. 等都会被下划线替代。符号名称中的 _binary 前缀由 objcopy 命令添加，对文本文件和二进制文件都是如此。

如果要将文件嵌入到项目中，而非组件中，可以调用 target_add_binary_data 函数：

target_add_binary_data(myproject.elf "main/data.bin" TEXT)

并将这行代码放在项目 CMakeLists.txt 的 project() 命令之后，修改 myproject.elf 为你自己的项目名。如果最后一个参数是 TEXT，那么构建系统会嵌入以 null 结尾的字符串，如果最后一个参数被设置为 BINARY，则将文件内容按照原样嵌入。

有关使用此技术的示例，请查看 file_serving 示例 protocols/http_server/file_serving/main/CMakeLists.txt 中的 main 组件，两个文件会在编译时加载并链接到固件中。

也可以嵌入生成的文件：

add_custom_command(OUTPUT my_processed_file.bin
  COMMAND my_process_file_cmd my_unprocessed_file.bin)

target_add_binary_data(my_target "my_processed_file.bin" BINARY)
上述示例中，`my_processed_file.bin` 是通过命令 `my_process_file_cmd` 从文件 `my_unprocessed_file.bin` 中生成，然后嵌入到目标中。

使用 `DEPENDS` 参数来指明对目标的依赖性:

```cpp
add_custom_target(my_process COMMAND ...) 
target_add_binary_data(my_target "my_embed_file.bin" BINARY DEPENDS my_process)
```

target_add_binary_data 的 `DEPENDS` 参数确保目标首先执行。

### 代码和数据的存放

ESP-IDF 还支持自动生成链接脚本，它允许组件通过链接片段文件定义其代码和数据在内存中的存放位置。构建系统会处理这些链接片段文件，并将处理后的结果扩充进链接脚本，从而指导应用程序二进制文件的链接过程。更多详细信息与快速上手指南，请参阅【链接脚本生成机制】。

### 完全覆盖组件的构建过程

当然，在有些情况下，上面提到的方法不一定够用。如果组件封装了另一个第三方组件，而这个第三方组件并不能直接在 ESP-IDF 的构建系统中工作。在这种情况下，就需要放弃 ESP-IDF 的构建系统，改为使用 CMake 的 `ExternalProject` 功能。组件 CMakeLists 示例如下:

```cpp
# 用于 quirc 的外部构建过程，在源目录中运行
# 并生成 libquirc.a
externalproject_add(quirc_build
    PREFIX ${COMPONENT_DIR}/quirc
    SOURCE_DIR ${COMPONENT_DIR}/quirc
    CONFIGURE_COMMAND "
    BUILD_IN_SOURCE 1
    BUILD_COMMAND make CC=${CMAKE_C_COMPILER} libquirc.a
    INSTALL_COMMAND "
)

# 将 libquirc.a 添加到构建系统中
add_library(quirc STATIC IMPORTED GLOBAL)
add_dependencies(quirc quirc_build)

set_target_properties(quirc PROPERTIES IMPORTED_LOCATION
    ${COMPONENT_DIR}/quirc/libquirc.a)
set_target_properties(quirc PROPERTIES INTERFACE_INCLUDE_DIRECTORIES
    ${COMPONENT_DIR}/quirc/lib)
set_directory_properties( PROPERTIES ADDITIONAL_MAKE_CLEAN_FILES
    "${COMPONENT_DIR}/quirc/libquirc.a")
```

（上述 CMakeLists.txt 可用于创建名为 quirc 的组件，该组件使用自己的 Makefile 构建 quirc 项目。）

- `externalproject_add` 定义了一个外部构建系统。
  - 设置 `SOURCE_DIR`、`CONFIGURE_COMMAND`、`BUILD_COMMAND` 和 `INSTALL_COMMAND`。如果外部构建系统没有配置这一步骤，可以将 `CONFIGURE_COMMAND` 设置为空字符串。在 ESP-IDF 的构建系统中，一般会将 `INSTALL_COMMAND` 变量设置为 `empty`。
  - 设置 `BUILD_IN_SOURCE`，即构建目录与源目录相同。否则，您也可以设置 `BUILD_DIR` 变量。
  - 有关 `externalproject_add()` 命令的详细信息，请参阅【ExternalProject】。
- 第二组命令添加了一个目标库，指向外部构建系统生成的库文件。为了添加 `include` 目录，并告知 CMake 该文件的位置，需要再设置一些属性。
- 最后，生成的库被添加到 `ADDITIONAL_MAKE_CLEAN_FILES` 中，即执行 `make clean` 后会删除该库。请注意，构建系统中的其他目标文件不会被删除。
Chapter 4. API 指南

备注：当外部构建系统使用 PSRAM 时，请记得将 `--mfix-esp32-psram-cache-issue` 添加到 C 编译器的参数中。关于该标志的更多详细信息，请参考 `CONFIG_SPIRAM_CACHE_WORKAROUND`。

ExternalProject 的依赖与构建清理 对于外部项目的构建，CMake 会有一些不同寻常的行为：

- **ADDITIONAL_MAKE_CLEAN_FILES** 仅在使用 Make 构建系统时有效。如果使用 Ninja 或 IDE 自带的构建系统，执行项目清理时，这些文件不会被删除。
- **ExternalProject** 会在 clean 运行后自动重新运行配置和构建命令。
- 可以采用以下两种方法来配置外部构建命令：
  1. 将外部 BUILD_COMMAND 命令设置为对所有源代码完整的重新编译。如果传递给 externalproject_add 命令的 DEFENDS 的依赖项发生了改变，或者当前执行的是项目清理操作（即运行了 ids.py clean, ninja clean 或者 make clean），那么就会执行该命令。
  2. 将外部 BUILD_COMMAND 命令设置为增量式构建命令，并给 externalproject_add 传递 BUILD_ALWAYS 1 参数。即不管实际的依赖情况，每次构建时，都会构建外部项目。这种方式仅当外部构建系统具有增量式构建的能力，且运行时间不会很长时才推荐。

构建外部项目的最佳方法取决于项目本身，其构建系统，以及是否需要频繁重新编译项目。

4.5.13 自定义 sdkconfig 的默认值

对于示例工程或者其他您不想指定完整 sdkconfig 配置的项目，但是您确实希望覆盖 ESP-IDF 默认值中的某些键值，则可以在项目中创建 sdkconfig.defaults 文件。重新创建新配置时将会用到此文件，另外在 sdkconfig 没有设置新配置值时，上述文件也会被用到。

如若需要覆盖此文件的名称或指定多个文件，请设置 **SDKCONFIG_DEFAULTS** 环境变量或在顶层 CMake-Lists.txt 文件中设置 **SDKCONFIG_DEFAULTS**。非绝对路径的文件名将以当前项目的相对路径来解析。

在指定多个文件时，使用分号作为分隔符。先列出的文件将会先应用。如果某个键值在多个文件里定义，后面文件的定义会覆盖前面文件的定义。

一些 IDF 示例中包含了 sdkconfig.ci 文件。该文件是 CI（持续集成）测试框架的一部分，在正常构建过程中会被忽略。

依赖于硬件目标的 sdkconfig 默认值

除了 sdkconfig.defaults 之外，构建系统还将从 sdkconfig.defaults.TARGET_NAME 文件加载默认值，其中 **IDF_TARGET** 的值为 TARGET_NAME。例如，对于 ESP32 这个硬件目标，sdkconfig 的默认值会首先从 sdkconfig.defaults 获取，然后再从 sdkconfig.defaults.esp32 获取。

如果使用 **SDKCONFIG_DEFAULTS** 覆盖默认文件的名称，则硬件目标的默认文件名也会从 **SDKCONFIG_DEFAULTS** 值中派生。如果 **SDKCONFIG_DEFAULTS** 中有多个文件，硬件目标文件会在引入该硬件目标文件的文件之后应用，而 **SDKCONFIG_DEFAULTS** 中所有其它后续文件则会在硬件目标文件之后应用。

例如，如果 **SDKCONFIG_DEFAULTS**="sdkconfig.defaults;sdkconfig_devkit1"，并且在同一文件夹中有一个 sdkconfig.defaults.esp32 文件，那么这些文件将按以下顺序应用：(1) sdkconfig.defaults (2) sdkconfig.defaults.esp32 (3) sdkconfig_devkit1。

4.5.14 Flash 参数

有些情况下，我们希望在没有 IDF 时也能烧写目标板，为此，我们希望可以保存已构建的二进制文件。esptool.py 和 esptool write flash 命令的参数。可以通过编写一段简单的脚本来保存二进制文件和 esptool.py。

运行项目构建之后，构建目录中包含项目二进制输出文件 (.bin 文件)，同时也包含以下烧录数据文件：

Espressif Systems 2145 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
• `flash_project_args` 包含烧录整个项目的参数，包括应用程序 (app)、引导程序 (bootloader)、分区表，如果设置了 PHY 数据，也会包含此数据。
• `flash_app_args` 只包含烧录应用程序的参数。
• `flash_bootloader_args` 只包含烧录引导程序的参数。

您可以参照如下命令将任意烧录参数文件传递给 esptool.py:

```sh
python esptool.py --chip esp32 write_flash @build/flash_project_args
```

也可以手动复制参数文件中的数据到命令行中执行。

构建目录中还包含生成的 flasher_args.json 文件，此文件包含 JSON 格式的项目烧录信息，可用于 `idf.py` 和其它需要项目构建信息的工具。

### 4.5.15 构建 Bootloader

引导程序是 `/components/bootloader/subproject` 内部独特的“子项目”，它有自己的项目 CMakeLists.txt 文件，能够构建独立于主项目的 .ELF 和 .BIN 文件，同时它又与主项目共享配置和构建目录。

子项目通过 `/components/bootloader/project_include.cmake` 文件作为外部项目插入到项目的顶层，主构建进程会运行子项目的 CMake，包括查找组件（主项目使用的组件的子集），生成引导程序专用的配置文件（从主 `sdkconfig` 文件中派生）。

### 4.5.16 编写纯 CMake 组件

ESP-IDF 构建系统用“组件”的概念“封装”了 CMake，并提供了很多帮助函数来自动将这些组件集成到项目构建当中。

然而，“组件”概念的背后是一个完整的 CMake 构建系统，因此可以制作纯 CMake 组件。

下面是使用纯 CMake 语法为 `json` 组件编写的最小 CMakeLists 文件的示例:

```cmake
add_library(json STATIC cJSON/cJSON.c cJSON/cJSON_Utils.c) target_include_directories(json PUBLIC cJSON)
```

• 这实际上与 IDF 中的 json 组件 是等效的。
• 因为组件中的源文件不多，所以这个 CMakeLists 文件非常简单，对于具有大量源文件的组件而言，ESP-IDF 支持的组件通配符，可以简化组件 CMakeLists 的样式。
• 每当组件中新增一个与组件同名的库目标时，ESP-IDF 构建系统会自动将其添加到构建中，并公开公共的 include 目录，如果组件想要添加一个与组件不同名的库目标，就需要使用 CMake 命令手动添加依赖关系。

### 4.5.17 组件中使用第三方 CMake 项目

CMake 在许多开源的 C/C++ 项目中广泛使用。用户可以在自己的应用程序中使用开源代码。CMake 构建系统的一大好处就是可以导入这些第三方的项目，有时候甚至不用做任何改动。这就允许用户使用当前 ESP-IDF 组件尚未提供的功能，或者使用其它库来实现相同的功能。

假设 main 组件需要导入一个假设库 foo，相应的组件 CMakeLists 文件如下所示:

```cmake
# 注册组件
idf_component_register(...) 
# 设置 ‘foo’ 项目中的一些 CMake 变量，以控制 ‘foo’ 的构建过程
set(FOO_BUILD_STATIC OFF) 
set(FOO_BUILD_TESTS OFF) 
```

（下页继续）
实际的案例请参考 build_system/cmake/import_lib。请注意，导入第三方库所需要做的工作可能会因库的不同而有所差异。建议详细阅读第三方库的文档，了解如何将其导入到其它项目中。阅读第三方库的 CMakeLists.txt 文件以及结构也会有所帮助。

用这种方式还可以将第三方库封装成 ESP-IDF 的组件。例如 mbedls 组件就是封装了 mbedls 项目得到的。详情请参考 mbedls 组件的 CMakeLists.txt 文件。

每当使用 ESP-IDF 构建系统时，CMake 变量 ESP_PLATFORM 都会被设置为 1。如果要在通用的 CMake 代码加入 IDF 特定的代码时，可以采用 if (ESP_PLATFORM) 的形式加以分隔。

**外部库中使用 ESP-IDF 组件**

上述示例中假设的是外部库 foo（或 import_lib 示例中的 tinyxml 库）除了常见的 API 如 libc、libstdc++ 等外不需要使用其它 ESP-IDF API。如果外部库需要使用其它 ESP-IDF 组件提供的 API，则需要在外部 CMakeLists.txt 文件中通过添加对库目标 idf::<componentname> 的依赖关系。

例如，在 foo/CMakeLists.txt 文件:

```cmake
add_library(foo bar.c fizz.cpp buzz.cpp)
if (ESP_PLATFORM)
    # 在 ESP-IDF 中， bar.c 需要包含 spi_flash 组件中的 esp_flash.h
    target_link_libraries(foo PRIVATE idf::spi_flash)
endif()
```

### 4.5.18 组件中使用预建库

还有一种情况是您有一个由其它构建过程生成预建静态库 (.a 文件)。

ESP-IDF 构建系统为用户提供了一个实用函数 add_prebuilt_library，能够轻松导入并使用预建库:

```cmake
add_prebuilt_library(target_name lib_path [REQUIRES req1 req2 ...] [PRIV_REQUIRES _ req1 req2 ...])
```

其中:

- **target_name** - 用于引用导入库的名称，如链接到其它目标时
- **lib_path** - 预建库的路径，可以是绝对路径或是相对于组件目录的相对路径

可选参数 REQUIRES 和 PRIV_REQUIRES 指定对于其它组件的依赖性，这些参数与 idf_component_register 的参数的意义相同。

注意预建库的编译目标需与预编的项目相同。预建库的相关参数也要匹配。如果不特别注意，这两个因素可能会导致应用程序中出现 bug。

请查看示例 build_system/cmake/import_prebuilt。

### 4.5.19 在自定义 CMake 项目中使用 ESP-IDF

ESP-IDF 提供了一个模板 CMake 项目，可以基于此轻松创建应用程序。然而在有些情况下，用户可能已有一个现成的 CMake 项目，或者想自己创建一个 CMake 项目，此时就希望将 IDF 中的组件以库的形式链接到用户目标（库/可执行文件）。
可以通过 tools/cmake/idf.cmake 提供的build system APIs 实现该目标。例如:

```cmake
# 导入 ESP- IDF CMake 构建系统 API 的 CMake 文件
include($ENV{IDF_PATH}/tools/cmake/idf.cmake)

# 在构建中导入 ESP-IDF 组件，可以视作等同 add_subdirectory()
# 但为 ESP-IDF 构建增加额外的构建过程
# 具体构建过程
idf_build_process(esp32)

# 创建项目可执行文件
# 使用其别名 idf::newlib 将其链接到 newlib 组件
add_executable(${CMAKE_PROJECT_NAME}.elf main.c)
target_link_libraries(${CMAKE_PROJECT_NAME}.elf idf::newlib)

# 让构建系统知道项目可执行文件是什么，从而添加更多的目标以及依赖关系等
idf_build_executable(${CMAKE_PROJECT_NAME}.elf)
```

build_system/cmake/idf_as_lib 中的示例演示了如何在自定义的 CMake 项目创建一个类似于 Hello World 的应用程序。

**备注:** IDF 构建系统只能为其构建的源文件设置编译器标志。当使用外部 CMakeLists.txt 文件并启用 PSRAM 时，记得在 C 编译器参数中添加 mfix-esp32-psram-cache-issue。参见 ref:CONFIG_SPIRAM_CACHE_WORKAROUND 了解更多信息。

### 4.5.20 ESP-IDF CMake 构建系统 API

**ESP-IDF 构建命令**

```cmake
idf_build_get_property(var property [GENERATOR_EXPRESSION])
```

该函数查找一个构建属性 property，并将其存储在当前作用域可视的 var 中。特定 GENERATOR_EXPRESSION 将检查该属性的生成器表达式字符串（不是实际值），它可与支持生成器表达式的 CMake 命令一起使用。

```cmake
idf_build_set_property(property val [APPEND])
```

该函数设置构建属性 property 的值为 val。特定 APPEND 将把指定的值附加到属性当前值之后。如果该属性之前不存在或当前为空，则指定的值将变为第一个元素/成员。

```cmake
idf_build_component(component_dir)
```

该函数向构建系统提交一个包含组件的 component_dir 目录。相对路径会被转换为相对于当前目录的绝对路径。所有对该命令的调用必须在 idf_build_process 之前执行。该命令并不保证组件在构建过程中会被处理（参见 idf_build_process 中 COMPONENTS 参数说明）

```cmake
idf_build_process(target
    [PROJECT_DIR project_dir]
    [PROJECT_VER project_ver]
    [PROJECT_NAME project_name]
    [SDKCONFIG sdkconfig]
    [SDKCONFIG_DEFAULTS sdkconfig_defaults]
    [BUILD_DIR build_dir]
    [COMPONENTS component1 component2 ...])
```
为了导人 ESP- IDF 组件执行大量的幕后工作，包括组件配置、库创建、依赖性扩展和解析。在这些功能中，对于用户最重要的可能是通过调用每个组件的 idf_component_register 来创建库。该命令为每个组件创建库；这些库可以使用别名来访问，其形式为 idf::component_name。这些别名可以用来将组件链接到用户自己的目标、库或可执行文件上。

该调用要求用 target 参数指定目标芯片。调用的可选参数包括：

- `PROJECT_DIR` - 项目目录，默认为 CMAKE_SOURCE_DIR。
- `PROJECT_NAME` - 项目名称，默认为 CMAKE_PROJECT_NAME。
- `PROJECT_VER` - 项目的版本/版本号，默认为“1”。
- `SDKCONFIG` - 生成的 sdkconfig 文件的输出路径。根据是否设置 PROJECT_DIR，默认为 PROJECT_DIR/sdkconfig 或 CMAKE_SOURCE_DIR/sdkconfig。
- `SDKCONFIG_DEFAULTS` - 包含默认配置的文件列表（列表中必须包含完整的路径）。默认为空；对于列表中的每一个值 filename，如果存在的话，也会加载文件 filename.target 中的配置。对于列表中的 filename 的每一个值，也会加载文件 filename.target（如果存在的话）中的配置。
- `BUILD_DIR` - 用于放置 ESP-IDF 构建相关工具的目录，如生成的二进制文件、文本文件、组件；默认为 CMAKE_BINARY_DIR。
- `COMPONENTS` - 从构建系统已知的组件中选择要处理的组件（通过 idf_build_component 添加）。这个参数用于精简构建过程。如果在依赖链中需要其它组件，则会自动添加，即自动添加这个列表中组件的公共和私有依赖项，进而添加这些依赖项的公共和私有依赖，以此类推。如果不指定，则会处理构建系统已知的所有组件。

```cpp
idf_build_executable(executable)
```

指定 ESP-IDF 构建的可执行文件 executable。这将添加额外的目标，如与 flash 相关的依赖关系，生成额外的二进制文件等。应在 idf_build_process 之后调用。

```cpp
idf_build_get_property(var config [GENERATOR_EXPRESSION])
```

获取指定配置的值。就像构建属性一样，特定 GENERATOR_EXPRESSION 将检索该配置的生成器表达式字符串，而不是实际值，可以与支持生成器表达式的 CMake 命令一起使用。然而，实际的配置值只有在调用 idf_build_process 后才能知道。

### ESP-IDF 构建属性

可以通过使用构建命令 idf_build_get_property 来获取构建属性的值。例如，以下命令可以获取构建过程中使用的 Python 解释器的相关信息。

```cpp
idf_build_get_property(python PYTHON)
message(STATUS "The Python interpreter is: \$\{python\}"
```

- `BUILD_DIR` - 构建目录：由 idf_build_process 的 BUILD_DIR 参数设置。
- `BUILD_COMPONENTS` - 包含在构建中的组件列表：由 idf_build_process 设置。
- `BUILD_COMPONENT_ALIASES` - 包含在构建中的组件的库别名列表：由 idf_build_process 设置。
- `C_COMPILE_OPTIONS` - 适用于所有组件的 C 源代码文件的编译选项。
- `COMPILER_OPTIONS` - 适用于所有组件的源文件（无论是 C 或是 C++）的编译选项。
- `COMPILER_DEFINITIONS` - 适用于所有组件源文件的编译定义。
- `CXX_COMPILE_OPTIONS` - 适用于所有组件的 C++ 源文件的编译选项。
- `EXECUTABLE` - 项目可执行文件；通过调用 idf_build_executable 设置。
- `EXECUTABLE_NAME` - 不含扩展名的项目可执行文件的名称；通过调用 idf_build_executable 设置。
- `EXECUTABLE_DIR` - 输出的可执行文件的路径。
- `IDF_COMPONENT_MANAGER` - 默认启用组件管理器，但如果设置这个属性为 ‘0’，则会被 IDF_COMPONENT_MANAGER 环境变量禁用。
- `IDF_PATH` - ESP-IDF 路径：由 IDFF PATH 环境变量设置，或者从 idf cmake 的位置推断。
- `IDF_TARGET` - 构建的目标芯片：由 idf_build_process 的目标参数设置。
- `IDF_VER` - ESP-IDF 版本：由版本文件或 IDF_PATH 仓库的 Git 版本设置。
- `INCLUDE_DIRECTORIES` - 包含所有组件源文件的目录。
• KCONFIGS - 构建过程中组件里的 Kconfig 文件的列表，由 idf_build_process 设置。
• KCONFIG_PROJBUILDS - 构建过程中的 Kconfig.projbuild 文件的列表，由 idf_build_process 设置。
• PROJECT_NAME - 项目的名称，由 idf_build_process 的 PROJECT_NAME 参数设置。
• PROJECT_DIR - 项目的目录，由 idf_build_process 的 PROJECT_DIR 参数设置。
• PROJECT_VER - 项目的版本，由 idf_build_process 的 PROJECT_VER 参数设置。
• PYTHON - 用于构建的 Python 解释器；如果有则从 PYTHON 环境变量中设置，如果没有，则使用“python”。
• SDKCONFIG - 输出的配置文件的完整路径：由 idf_build_process SDKCONFIG 参数设置。
• SDKCONFIG_DEFAULTS - 包含默认配置的文件列表：由 idf_build_process SDKCONFIG_DEFAULTS 参数设置。
• SDKCONFIG_HEADER - 包含组件配置的 C/C++ 头文件的完整路径：由 idf_build_process 设置。
• SDKCONFIG_CMAKE - 包含组件配置的 CMake 文件的完整路径：由 idf_build_process 设置。
• SDKCONFIG_JSON - 包含组件配置的 JSON 文件的完整路径：由 idf_build_process 设置。
• SDKCONFIG_JSON_MENUS - 包含配置菜单的 JSON 文件的完整路径：由 idf_build_process 设置。

ESP-IDF 组件命令

```cpp
didf_component_get_property(var component property [GENERATOR_EXPRESSION])
```

检索一个指定的 component 的组件属性 property，并将其存储在当前作用域可访问的 var 中。指定 GENERATOR_EXPRESSION 将检索该属性的生成器表达式字符串（不是实际值），它可以在支持生成器表达式的 CMake 命令中使用。

```cpp
didf_component_set_property(component property val [APPEND])
```

设置指定的 component 的组件属性，property 的值为 val。特APPEND 将把指定的值追加到属性的当前值后。如果该属性之前不存在或当前为空，指定的值将成为第一个元素/成员。

```cpp
didf_component_register([[[SRCS src1 src2 ...] | [[SRC_DIRS dir1 dir2 ...]] [EXCLUDE_\n ->SRCS src1 src2 ...]]
 [INCLUDE_DIRS dir1 dir2 ...]
 [PRIV_INCLUDE_DIRS dir1 dir2 ...]
 [REQUIRES component1 component2 ...]
 [PRIV_REQUIRES component1 component2 ...]
 [LDFRAGMENTS ldfragment1 ldfragment2 ...]
 [REQUIRED_IDF_TARGETS target1 target2 ...]
 [MBED_FILES file1 file2 ...]
 [MBED_TXTFILES file1 file2 ...]
 [CONFIG kconfig]
 [KCONFIG_PROJBUILD kconfigprojbuild]
 [WHOLE_ARCHIVE])
```

将一个组件注册到构建系统中，就像 project() CMake 命令一样，该命令应该直接从组件的 CMakeLists.txt 中调用（而不是通过函数或宏），且建议在其他命令之前调用该命令。下面是一些关于在 idf_component_register 之前不能调用哪些命令的指南：

- 在 CMake 脚本模式下无效的命令。
- 在 project() include.cmake 中定义的自定义命令。
- 除了 idf_build_get_property 之外，构建系统的 API 命令；但要考虑该属性是否有被设置。

对变量进行设置和操作的命令，一般可在 idf_component_register 之前调用。

idf_component_register 的参数包括：

- SRCS - 组件的源文件，用于为组件创建静态库；如果没有指定，组件将被视为仅配置组件，从而创建接口库。
Chapter 4. API

- **SRC_DIRS, EXCLUDE_SRCS** - 用于通过指定目录来 glob 源文件 (.c, .cpp, .S)。而不是通过 SRCS 手动指定源文件。请注意，这受 `CMake` 中指定配置的限制。在 `EXCLUDE_SRCS` 中指定的源文件会从被 glob 的文件中移除。
- **INCLUDE_DIRS** - 相对于组件目录的路径，该路径将被添加到需要当前组件的所有其他组件的 include 搜索路径中。
- **PRIV_INCLUDE_DIRS** - 必须是相对于组件目录的目录路径，它仅被添加到这个组件源文件的 include 搜索路径中。
- **REQUIRES** - 组件的公共组件依赖项。
- **PRIV_REQUIRES** - 组件的私有组件依赖项：在仅用于配置的组件上会被忽略。
- **LDFRAGMENTS** - 组件链接器片段文件。
- **REQUIRED_IDF_TARGETS** - 指定该组件唯一支持的目标。
- **KCONFIG** - 覆盖默认的 `Kconfig` 文件。
- **KCONFIG_PROJBUILD** - 覆盖默认的 `Kconfig.projbuild` 文件。
- **WHOLE_ARCHIVE** - 如果指定了此参数，链接时会在组件库的前后分别添加 `-Wl,--whole-archive` 和 `-Wl,--nowhole-archive`。这与设置 `WHOLE_ARCHIVE` 组件属性的效果一致。

以下内容用于将数据嵌入到组件中，并在确定组件是否仅用于配置时被视为源文件。这意味着，即使组件没有指定源文件，如果组件指定了以下其中之一，仍然会在内部为组件创建一个静态库。

- **EMBED_FILES** - 嵌入组件的二进制文件
- **EMBED_TXTFILES** - 嵌入组件的文本文件

**ESP-IDF 组件属性**

组件的属性值可以通过使用构建命令 `idf_component_get_property` 来获取。例如，以下命令可以获取 `freertos` 组件的目标。

```bash
idf_component_get_property(dir freertos COMPONENT_DIR)
message({
STATUS "The 'freertos' component directory is: \${dir}"}
```

- **COMPONENT_ALIAS** - `COMPONENT_LIB` 的别名，用于将组件链接到外部目标：由 `idf_build_component` 设置，别名库本身由 `idf_component_register` 创建。
- **COMPONENT_DIR** - 组件目录：由 `idf_build_component` 设置。
- **COMPONENT_OVERRIDEN_DIR** - 如果这个组件覆盖了另一个组件，则包含原组件的目录。
- **COMPONENT_LIB** - 所创建的组件静态/可执行的库名称：由 `idf_build_component` 设置，库本身由 `idf_component_register` 创建。
- **COMPONENT_NAME** - 组件的名称：由 `idf_build_component` 根据组件的目录名设置。
- **COMPONENT_TYPE** - 组件的类型（LIBRARY 或 CONFIG_ONLY）。如果一个组件指定了源文件或嵌入了一个文件，那么它的类型就是 LIBRARY。
- **EMBED_FILES** - 要嵌入组件的文件列表：由 `idf_component_register EMBED_FILES` 参数设置。
- **EMBED_TXTFILES** - 要嵌入组件的文本文件列表：由 `idf_component_register EMBED_TXTFILES` 参数设置。
- **INCLUDE_DIRS** - 组件 include 目录列表：由 `idf_component_register INCLUDE_DIRS` 参数设置。
- **KCONFIG** - 组件 Kconfig 文件：由 `idf_build_component` 设置。
- **KCONFIG_PROJBUILD** - 组件 Kconfig.projbuild：由 `idf_build_component` 设置。
- **LDFRAGMENTS** - 组件链接器片段文件列表：由 `idf_component_register LDFRAGMENTS` 参数设置。
- **MANAGED_PRIV_REQUIRES** - IDF 组件管理器从 `idf_component.yml` 清单文件中的依赖关系中添加的私有组件依赖关系列表。
- **MANAGED_REQUIRES** - IDF 组件管理器从 `idf_component.yml` 清单文件的依赖关系中添加的公共组件依赖关系列表。
- **PRIV_INCLUDE_DIRS** - 组件私有 include 目录列表：在 LIBRARY 类型的组件 `idf_component_register PRIV_INCLUDE_DIRS` 参数中设置。
- **PRIV_REQUIRES** - 私有组件依赖关系列表：根据 `idf_component_register PRIV_REQUIRES` 参数的值以及 `idf_component.yml` 清单文件中的依赖关系设置。
• REQUIRED_IDF_TARGETS - 组件支持的目标列表：由 `idf_component_register` EM-BED_TXTFILES 参数设置。
• REQUIREES - 公共组件依赖关系列表：根据 `idf_component_register` REQUIREES 参数的值以及 `idf_component.yaml` 清单文件中的依赖关系设置。
• SRCS - 组件源文件列表：由 `idf_component_register` 的 SRCS 或 SRC_DIRS/EXCLUDE_SRCS 参数设置。
• WHOLE_ARCHIVE - 如果该属性被设置为 TRUE (或是其他 CMake 布尔“真”值：1,ON,YES,Y 等),链接时会在组件库的前后分别添加-Wl,--whole-archive 和 -Wl,--no-whole-archive 选项。这可以强制链接器将每个目标文件包含到可执行文件中，即使该目标文件没有解析来自应用程序其余部分的任何引用。当组件中包含依赖链接时注册的插件或模块时，通常会使用该方法。默认情况下，此属性为 FALSE。可以从组件的 CMakeLists.txt 文件中将其设置为 TRUE。

### 4.5.21 文件遍历 & 增量构建

在 ESP-IDF 组件中添加源文件的首选方法是在 COMPONENT_SRCS 手动列出输出它们:

```bash
idf_component_register(SRCS library/a.c library/b.c platform/platform.c ...)
```

这是在 CMake 中手动列出源文件的 最佳实践。然而，当有许多源文件都需要添加到构建中时，这种方法就会很不方便。ESP-IDF 构建系统因此提供了另一种替代方法，即使用 SRC_DIRS 来指定源文件:

```bash
idf_component_register(SRC_DIRS library platform ...
```

后台会使用遍历在指定的目录中查找源文件。但是请注意，在使用这种方法的时候，如果组件中添加了一个新的源文件，CMake 并不知道重新运行配置，最终该文件也没有被加入构建中。

如果是自己添加的源文件，这种折衷还是可以接受的，因为用户可以触发一次干净的构建，或者运行 `idf.py reconfigure` 来手动重启 CMake。但是，如果你需要与其它使用 Git 等版本控制工具的开发人员共享项目时，问题就会变得非常困难，因为开发人员有可能会拉取新的版本。

ESP-IDF 中的组件使用了第三方的 Git CMake 集成模块（/tools/cmake/third_party/GetGitRevisionDescription.cmake），任何时候源码仓库的提交记录发生了改变，该模块就会自动重新运行 CMake。即只要拉取了新的 ESP-IDF 版本，CMake 就会重新运行。

对于不属于 ESP-IDF 的项目组件，有以下几个选项供参考:

- 如果项目文件保存在 Git 中，ESP-IDF 会自动跟踪 Git 修订版本，并在它发生变化时重新运行 CMake。
- 如果一些组件保存在第三方 Git 仓库中（不在项目仓库或 ESP-IDF 仓库），则可以在组件 CMakeLists 文件中调用 `git_describe` 函数，以便在 Git 修订版本发生变化时自动重启 CMake。
- 如果没有使用 Git，请记住在源文件发生变化时手动运行 `idf.py reconfigure`。
- 使用 `idf_component_register` 的 SRCS 参数来列出项目组件中的所有源文件则可以完全避免这一问题。

具体选择哪一方式，就要取决于项目本身，以及项目用户。

### 4.5.22 构建系统的元数据

为了将 ESP-IDF 集成到 IDE 或者其它构建系统中，CMake 构建过程中会在 build/ 目录下生成大量元数据文件。运行 `cmake` 或 `idf.py reconfigure` (或任何其它 `idf.py` 构建命令), 可以重新生成这些元数据文件。

- `compile_commands.json` 是标准格式的 JSON 文件，它描述了在项目中参与编译的每个源文件。
- `CMake` 其中的一个功能就是生成此文件，许多 IDE 都知道如何解析此文件。
- `project_description.json` 包含有关 ESP-IDF 项目、已配置路径等的一些常规信息。
- `flasher_args.json` 包含 `esptool.py` 工具用于烧录项目二进制文件的参数，此外还有 `flasherargs` 文件，可直接与 `esptool.py` 一起使用。更多详细信息请参阅 `Flash 参数`。
- `CMakeCache.txt` 是 CMake 的缓存文件，包含 CMake 进程、工具链等其它信息。
• config/sdkconfig.json 包含 JSON 格式的项目配置结果。
• config/kconfig_menus.json 是在 menuconfig 中显示菜单的 JSON 格式版本，用于外部 IDE 的 UI。

JSON 配置服务器

kconfserver 工具可以帮助 IDE 轻松地与配置系统的逻辑进行集成，它运行在后台，通过使用 stdin 和 stdout 读写 JSON 文件的方式与调用进程交互。

您可以通过 idf.py confserver 或 ninja kconfserver 从项目中运行 kconfserver，也可以使用不同的构建生成器来触发类似的目标。

有关 kconfserver 的更多信息，请参阅 esp-idf-kconfig documentation.

4.5.23 构建系统内部

构建脚本

ESP-IDF 构建系统的列表文件位于 /tools/cmake 中。实现构建系统核心功能的模块如下
• build.cmake - 构建相关命令，即构建初始化、检查/设置构建属性、构建处理。
• component.cmake - 组件相关的命令，如添加组件、检查/设置组件属性、注册组件。
• kconfig.cmake - 从 Kconfig 文件中生成配置文件（sdkconfig、sdkconfig.h、sdkconfig.cmake 等）。
• ldgen.cmake - 从链接器片段文件生成最终链接器脚本。
• target.cmake - 设置构建目标和工具链文件。
• utilities.cmake - 其它帮助命令。

除了这些文件，还有两个重要的 CMake 脚本在 /tools/cmake 中：
• idf.cmake - 设置构建参数并导入上面列出的核心模块。之所以包括在 CMake 项目中，是为了方便访问 ESP-IDF 构建系统功能。
• project.cmake - 导入 idf.cmake，并提供了一个自定义的 "project()" 命令，该命令负责处理建立可执行文件时所有的繁重工作。包含在标准 ESP-IDF 项目的顶层 CMakeLists.txt 中。

/tools/cmake 中的其它文件都是构建过程中的支持性文件或第三方脚本。

构建过程

本节介绍了标准的 ESP-IDF 应用构建过程。构建过程可以大致分为四个阶段：

图 3: ESP-IDF Build System Process

初始化

该阶段为构建设置必要的参数。
• 在将 idf.cmake 导入 project.cmake 后，将执行以下步骤：
  - 在环境变量中设置 IDF_PATH 或从顶层 CMakeLists.txt 中包含的 project.
cmake 路径推断相对路径。
将/Tools/cmake添加到CMAKE_MODULE_PATH中，并导入核心模块和各种辅助第三方脚本。
- 设置构建工具/可执行文件，如默认的Python解释器。
- 获取ESP-IDF git修订版，并存储为IDF_VER。
- 设置全局构建参数，即编译选项、编译定义，包括所有组件的include目录。
- 将components中的组件添加到构建中。

*自定义project()命令的初始部分执行以下步骤：*
- 在环境变量或CMake缓存中设置IDF_TARGET以及设置相应要使用的“CMAKE_TOOLCHAIN_FILE”。
- 添加EXTRA_COMPONENTS_DIRS中的组件至构建中。
- 从COMPONENTS/EXCLUDE_COMPONENTS、SDKCONFIG、SDKCONFIG_DEFAULTS等变量中为调用命令idf_build_process()准备参数。

调用idf_build_process()命令标志着这个阶段的结束。

**枚举**
这个阶段会建立一个需要在构建过程中处理的组件列表。该阶段在idf_build_process()的前半部分进行。
- 检索每个组件的公共和私有依赖。创建一个子进程，以脚本模式执行每个组件的CMakeLists.txt。idf_component_register.REQUIRES和PRIV_REQUIRES参数的值会返回给父进程。这就是所谓的早期扩展。在这一过程中定义变量CMAKE_BUILD_EARLY_EXPANSION。
- 根据公共和私有的依赖关系，递归地建立各个组件。

**处理**
该阶段处理构建中的组件，是idf_build_process()的后半部分。
- 从sdkconfig文件中加载项目配置，并生成sdkconfig.cmake和sdkconfig.h头文件。这两个文件分别定义了可以从中构建脚本和C/C++源文件/头文件中访问的配置变量宏。
- 导入各组件的project_include.cmake。
- 将每个组件添加为一个子目录，处理其CMakeLists.txt。组件CMakeLists.txt调用注册命令idf_component_register添加源文件、导入目录、构建组件库、链接依赖关系等。

**完成**
该阶段是idf_build_process()剩余的步骤。
- 创建可执行文件并将其链接到组件库中。
- 生成project_description.json等项目元数据文件并且显示所建项目等相关信息。

请参考/Tools/cmake/project.cmake获取更多信息。

### 4.5.24 从ESP-IDF GNU Make构建系统迁移到CMake构建系统

ESP-IDF CMake构建系统与旧版的GNU Make构建系统在某些方面非常相似，开发者都需要提供include目录、源文件等。然而，有一个语法上的区别，即对于ESP-IDF CMake构建系统，开发者需要将这些作为参数传递给注册命令idf_component_register。

**自动转换工具**

在ESP-IDF v4.x版本中，tools/cmake/convert_to_cmake.py提供了项目自动转换工具。由于该脚本依赖于make构建系统，所以v5.0版本中不包含该脚本。
Chapter 4. API

CMake 中不可用的功能

有些功能已从 CMake 构建系统中移除，或者已经发生很大改变。GNU Make 构建系统中的以下变量已从 CMake 构建系统中删除:

- COMPONENT_BUILD_DIR: 由 CMAKE_CURRENT_BINARY_DIR 替代。
- COMPONENT_LIBRARY: 默认为 $(COMPONENT_NAME).a 但是库名可以被组件覆盖。在 CMake 构建系统中，组件库名称不可再被组件覆盖。
- CC, LD, AR, OBJCOPY: gcc xtensa 交叉工具链中每个工具的完整路径。CMake 使用 CMAKE_C_COMPILER, CMAKE_C_LINK_EXECUTABLE 和 CMAKE_OBJCOPY 进行替代。完整列表请参阅 CMake 语言变量。
- HOSTCC, HOSTLD, HOSTAR: 宿主机本地工具链中每个工具的全名。CMake 系统不再提供此变量，外部项目需要手动检测所需的宿主机工具链。
- COMPONENT_ADD_LDFLAGS: 用于覆盖链接标志。CMake 中使用 target_link_link libraries 命令替代。
- COMPONENT_ADD_LINKER_DEPS: 链接过程依赖的文件列表。target_link_link libraries 通常会自动推断这些依赖。对于链接脚本，可以使用自定义的 CMake 函数 target_linker_scripts。
- COMPONENT_ADD_SUBMODULES: 不再使用。CMake 会自动枚举 ESP-IDF 仓库中所有的子模块。
- COMPONENT_EXTRA_INCLUDES: 曾是 COMPONENT_PRIV_INCLUDEDIRS 变量的替代版本，仅支持绝对路径。CMake 系统中统一使用 COMPONENT_PRIV_INCLUDEDIRS（可以是相对路径，也可以是绝对路径）。
- COMPONENT_OBJS: 以前，可以以目标文件列表的方式指定组件源。现在，可以通过 COMPONENT_SRCS 以源文件列表的形式指定组件源。
- COMPONENT_OBJEXCLUDE: 已被 COMPONENT_SRCEXCLUDE 替换。用于指定源文件（绝对路径或组件目录的相对路径）。
- COMPONENT_EXTRA_CLEAN: 已被 ADDITIONAL_MAKE_CLEAN_FILES 属性取代，注意，CMake 对此项功能有部分限制。
- COMPONENT_OWNBUILTARGET & COMPONENT_OWNCLEANTARGET: 已被 CMake 外部项目 <ExternalProject> 替代，详细内容请参阅完全覆盖组件的构建过程。
- COMPONENT_CONFIGONLY: 已被 register_config_only_component() 函数替代，请参阅配置组件。
- CFLAGS, CPPFLAGS, CXXFLAGS: 已被相应的 CMake 命令替代，请参阅组件编译控制。

无默认值的变量

以下变量不再具有默认值:

- 源目录（Make 中的 COMPONENT_SRCDIRS 变量，CMake 中 idf_component_register 的 SRC_DIRS 参数）
- include 目录（Make 中的 COMPONENT_ADD_INCLUDEDIRS 变量，CMake 中 idf_component_register 的 INCLUDE_DIRS 参数）

不再需要的变量

在 CMake 构建系统中，如果设置了 COMPONENT_SRCS，就不需要再设置 COMPONENT_SRCDIRS。实际上，CMake 构建系统中如果设置了 COMPONENT_SRCDIRS，那么 COMPONENT_SRCS 就会被忽略。

从 Make 中烧录

仍然可以使用 make flash 或者类似的目标来构建和烧录，但是项目 sdkconfig 再不能用来指定串口和波特率。可以使用环境变量来覆盖串口和波特率的设置，详情请参阅使用 Ninja/Make 来烧录。

4.6 Core Dump
4.6.1 Overview

ESP-IDF provides support to generate core dumps on unrecoverable software errors. This useful technique allows post-mortem analysis of software state at the moment of failure. Upon the crash system enters panic state, prints some information and halts or reboots depending configuration. User can choose to generate core dump in order to analyse the reason of failure on PC later on. Core dump contains snapshots of all tasks in the system at the moment of failure. Snapshots include tasks control blocks (TCB) and stacks. So it is possible to find out what task, at what instruction (line of code) and what callstack of that task lead to the crash. It is also possible dumping variables content on demand if previously attributed accordingly. ESP-IDF provides special commands to help users to retrieve and analyse core dumps:

- `idf.py coredump-info` - prints crashed task’s registers, callstack, list of available tasks in the system, memory regions and contents of memory stored in core dump (TCBs and stacks)
- `idf.py coredump-debug` - creates core dump ELF file and runs GDB debug session with this file. User can examine memory, variables and tasks states manually. Note that since not all memory is saved in core dump only values of variables allocated on stack will be meaningful

For more information about core dump internals see the - Core dump internals

4.6.2 Configurations

There are a number of core dump related configuration options which user can choose in project configuration menu (`idf.py menuconfig`).

**Core dump data destination (Components -> Core dump -> Data destination)**

- Save core dump to Flash (Flash)
- Print core dump to UART (UART)
- Disable core dump generation (None)

**Core dump data format (Components -> Core dump -> Core dump data format)**

- ELF format (Executable and Linkable Format file for core dump)
- Binary format (Basic binary format for core dump)

The ELF format contains extended features and allow to save more information about broken tasks and crashed software but it requires more space in the flash memory. This format of core dump is recommended for new software designs and is flexible enough to extend saved information for future revisions. The Binary format is kept for compatibility reasons, it uses less space in the memory to keep data and provides better performance.

**Core dump data integrity check (Components -> Core dump -> Core dump data integrity check)**

- Use CRC32 for core dump integrity verification
- Use SHA256 for core dump integrity verification (only work in ELF format)

The CRC32 option provides better calculation performance and consumes less memory for storage. The SHA256 hash algorithm provides greater probability of detecting corruption than a CRC32 with multiple bit errors.

**Maximum number of tasks snapshots in core dump (Components -> Core dump -> Maximum number of tasks)**

**Delay before core dump is printed to UART (Components -> Core dump -> Delay before print to UART)**

The value is in ms.

**Handling of UART core dumps in IDF Monitor (Components -> Core dump -> Delay before print to UART)**

The value is base64 encoded.

- Decode and show summary (info_corefile)
- Don’t decode
Reserved stack size (Components -> Core dump -> Reserved stack size)

Size of the memory to be reserved for core dump stack. If 0 core dump process will run on the stack of crashed task/ISR, otherwise special stack will be allocated. To ensure that core dump itself will not overflow task/ISR stack set this to the value above 800.

4.6.3 Save core dump to flash

When this option is selected core dumps are saved to special partition on flash. When using default partition table files which are provided with ESP-IDF it automatically allocates necessary space on flash. But if users want to use its own layout file together with core dump feature it should define separate partition for core dump as it is shown below:

```
# Name, Type, SubType, Offset, Size
# Note: if you have increased the bootloader size, make sure to update the offsets...
→ to avoid overlap
nvs, data, nvs, 0x9000, 0x6000
phy_init, data, phy, 0xf000, 0x1000
factory, app, factory, 0x10000, 1M
coredump, data, coredump, 64K
```

There are no special requirements for partition name. It can be chosen according to the user application needs, but partition type should be ‘data’ and sub-type should be ‘coredump’. Also when choosing partition size note that core dump data structure introduces constant overhead of 20 bytes and per-task overhead of 12 bytes. This overhead does not include size of TCB and stack for every task. So partition size should be at least 20 + max tasks number x (12 + TCB size + max task stack size) bytes.

The example of generic command to analyze core dump from flash is:

```
idf.py coredump-info
```

or

```
idf.py coredump-debug
```

4.6.4 Print core dump to UART

When this option is selected base64-encoded core dumps are printed on UART upon system panic. In this case users should save core dump text body to some file manually and then run the following command:

```
idf.py coredump-info -c </path/to/saved/base64/text>
```

or

```
idf.py coredump-debug -c </path/to/saved/base64/text>
```

Base64-encoded body of core dump will be between the following header and footer:

```
== CORE DUMP START ==
<body of base64-encoded core dump, save it to file on disk>
== CORE DUMP END ==
```

The **CORE DUMP START** and **CORE DUMP END** lines must not be included in core dump text file.

4.6.5 ROM Functions in Backtraces

It is possible situation that at the moment of crash some tasks or/and crashed task itself have one or more ROM functions in their callstacks. Since ROM is not part of the program ELF it will be impossible for GDB to parse such callstacks, because it tries to analyse functions’ prologues to accomplish that. In that case callstack printing will be
broken with error message at the first ROM function. To overcome this issue, ROM ELF provided by Espressif is loaded automatically based on the target and its revision. More details about ROM ELFs can be found here.

### 4.6.6 Dumping variables on demand

Sometimes you want to read the last value of a variable to understand the root cause of a crash. Core dump supports retrieving variable data over GDB by attributing special notations declared variables.

**Supported notations and RAM regions**

- `COREDUMP_DRAM_ATTR` places variable into DRAM area which will be included into dump.
- `COREDUMP_RTC_ATTR` places variable into RTC area which will be included into dump.
- `COREDUMP_RTC_FAST_ATTR` places variable into RTC_FAST area which will be included into dump.

**Example**

1. In *Project Configuration Menu*, enable `COREDUMP TO FLASH`, then save and exit.
2. In your project, create a global variable in DRAM area as such as:

   ```
   // uint8_t global_var;
   COREDUMP_DRAM_ATTR uint8_t global_var;
   ```

3. In main application, set the variable to any value and `assert(0)` to cause a crash.

   ```
   global_var = 25;
   assert(0);
   ```

4. Build, flash and run the application on a target device and wait for the dumping information.
5. Run the command below to start core dumping in GDB, where `PORT` is the device USB port:

   ```
   idf.py coredump-debug
   ```

6. In GDB shell, type `p global_var` to get the variable content:

   ```
   (gdb) p global_var
   $1 = 25 '\031'
   ```

### 4.6.7 Running `idf.py coredump-info` and `idf.py coredump-debug`

`idf.py coredump-info --help` and `idf.py coredump-debug --help` commands can be used to get more details on usage.

**Related Documents**

Anatomy of core dump image  Core dump component can be configured to use old legacy binary format or the new ELF one. The ELF format is recommended for new designs. It provides more information about the CPU and memory state of a program at the moment when panic handler is entered. The memory state embeds a snapshot of all tasks mapped in the memory space of the program. The CPU state contains register values when the core dump has been generated. Core dump file uses a subset of the ELF structures to register these information. Loadable ELF segments are used for the memory state of the process while ELF notes (ELF.PT_NOTE) are used for process metadata (pid, registers, signal, …). Especially, the CPU status is stored in a note with a special name and type (CORE, NT_PRSTATUS type).

Here is an overview of coredump layout:
Chapter 4. API

4: Coredump ELF image format

- ELF formatted core dump image
  - Core dump image header: Contains information about core dump image such as image size, image version, number of tasks, etc.
  - ELF file header: Standard ELF header for core dump image.
  - ELF Segment Headers
    - Segment #1 Header: ELF headers for loadable and read segments showed in the data area.
    - Segment #2 ... Segment #N-1 Headers:

- ELF Segment Data (Tasks Data)
  - RT_PRSTATUS data for Task #1 ... Task #N
  - Task #1 TCB data, stack data
  - Task #2 ... Task #N-1 TCB data, stack data

- ESP core dump information segment data
  - Version control information
  - Extra information

- Core dump image checksum (CRC32 or SHA256)

图 4: Core dump ELF image format

5: Coredump binary image format

- Binary core dump image format
  - Core dump header: Describes basic information such as image size, image version, number of tasks, etc.
  - Task #1 image
  - Task #2 ... Task #N-1 image
  - Task #N image
  - Image checksum (CRC32)

- Represents task data for each task:
  - Task header: TCB address, stack start address, stack end address
  - Task data: Dumped TCB memory, dumped task stack

图 5: Core dump binary image format
Note: The format of image file showed on the above pictures represents current version of image and can be changed in future releases.

**Overview of implementation**

The figure below describes some basic aspects related to implementation of core dump:

![Diagram](image)

### 4.7 Deep Sleep Wake Stubs

ESP32 supports running a “deep sleep wake stub” when coming out of deep sleep. This function runs immediately as soon as the chip wakes up - before any normal initialisation, bootloader, or ESP-IDF code has run. After the wake stub runs, the SoC can go back to sleep or continue to start ESP-IDF normally.

Deep sleep wake stub code is loaded into “RTC Fast Memory” and any data which it uses must also be loaded into RTC memory. RTC memory regions hold their contents during deep sleep.

#### 4.7.1 Rules for Wake Stubs

Wake stub code must be carefully written:

- As the SoC has freshly woken from sleep, most of the peripherals are in reset states. The SPI flash is unmapped.
- The wake stub code can only call functions implemented in ROM or loaded into RTC Fast Memory (see below.)
- The wake stub code can only access data loaded in RTC memory. All other RAM will be uninitialised and have random contents. The wake stub can use other RAM for temporary storage, but the contents will be overwritten when the SoC goes back to sleep or starts ESP-IDF.
• RTC memory must include any read-only data (.rodata) used by the stub.
• Data in RTC memory is initialised whenever the SoC restarts, except when waking from deep sleep. When waking from deep sleep, the values which were present before going to sleep are kept.
• Wake stub code is a part of the main esp-idf app. During normal running of esp-idf, functions can call the wake stub functions or access RTC memory. It is as if these were regular parts of the app.

4.7.2 Implementing A Stub

The wake stub in esp-idf is called `esp_wake_deep_sleep()`. This function runs whenever the SoC wakes from deep sleep. There is a default version of this function provided in esp-idf, but the default function is weak-linked so if your app contains a function named `esp_wake_deep_sleep()` then this will override the default.

If supplying a custom wake stub, the first thing it does should be to call `esp_default_wake_deep_sleep()`. It is not necessary to implement `esp_wake_deep_sleep()` in your app in order to use deep sleep. It is only necessary if you want to have special behaviour immediately on wake.

If you want to swap between different deep sleep stubs at runtime, it is also possible to do this by calling the `esp_set_deep_sleep_wake_stub()` function. This is not necessary if you only use the default `esp_wake_deep_sleep()` function.

All of these functions are declared in the `esp_sleep.h` header under components/esp32.

4.7.3 Loading Code Into RTC Memory

Wake stub code must be resident in RTC Fast Memory. This can be done in one of two ways.

The first way is to use the `RTC_IRAM_ATTR` attribute to place a function into RTC memory:

```c
void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    // Add additional functionality here
}
```

The second way is to place the function into any source file whose name starts with `rtc_wake_stub`. Files names `rtc_wake_stub*` have their contents automatically put into RTC memory by the linker.

The first way is simpler for very short and simple code, or for source files where you want to mix “normal” and “RTC” code. The second way is simpler when you want to write longer pieces of code for RTC memory.

4.7.4 Loading Data Into RTC Memory

Data used by stub code must be resident in RTC memory.

The data can be placed in RTC Fast memory or in RTC Slow memory which is also used by the ULP.

Specifying this data can be done in one of two ways:

The first way is to use the `RTC_DATA_ATTR` and `RTC_RODATA_ATTR` to specify any data (writeable or read-only, respectively) which should be loaded into RTC memory:

```c
RTC_DATA_ATTR int wake_count;

void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    static RTC_RODATA_ATTR const char fmt_str[] = "Wake count $d\n";
    esp_rom_printf(fmt_str, wake_count++);
}
```
The RTC memory area where this data will be placed can be configured via menuconfig option named CONFIG_ESP32_RTCDATA_IN_FAST_MEM. This option allows to keep slow memory area for ULP programs and once it is enabled the data marked with RTC_DATA_ATTR and RTC_RODATA_ATTR are placed in the RTC fast memory segment otherwise it goes to RTC slow memory (default option). This option depends on the CONFIG_FREERTOS_UNICORE because RTC fast memory can be accessed only by PRO_CPU.

The attributes RTC_FAST_ATTR and RTC_SLOW_ATTR can be used to specify data that will be force placed into RTC_FAST and RTC_SLOW memory respectively. Any access to data marked with RTC_FAST_ATTR is allowed by PRO_CPU only and it is responsibility of user to make sure about it.

Unfortunately, any string constants used in this way must be declared as arrays and marked with RTC_RODATA_ATTR, as shown in the example above.

The second way is to place the data into any source file whose name starts with rtc_wake_stub.

For example, the equivalent example in rtc_wake_stub_counter.c:

```c
int wake_count;

void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    esp_rom_printf("Wake count \%d\n", wake_count++);
}
```

The second way is a better option if you need to use strings, or write other more complex code.

To reduce wake-up time use the CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP Kconfig option, see more information in Fast boot from Deep Sleep.

### 4.7.5 CRC Check For Wake Stubs

During deep sleep, all RTC Fast memory areas will be validated with CRC. When ESP32 wakes up from deep sleep, the RTC fast memory will be validated with CRC again. If the validation passes, the wake stubs code will be executed. Otherwise, the normal initialization, bootloader and esp-idf codes will be executed.

| 备注：When the CONFIG_ESP_SYSTEM_ALLOW_RTC_FAST_MEM_AS_HEAP option is enabled, all the RTC fast memory except the wake stubs area is added to the heap. |

## 4.8 错误处理

### 4.8.1 概述

在应用程序开发中，及时发现并处理在运行时期的错误，对于保证应用程序的健壮性非常重要。常见的运行时错误有如下几种：

- 可恢复的错误：
  - 通过函数的返回值（错误码）表示的错误
  - 使用throw关键字抛出的C++异常
- 不可恢复（严重）的错误：
  - 断言失败（使用assert宏或者其它类似方法，可参考 Assertions）或者直接调用abort()函数造成的错误
  - CPU异常：访问受保护的内存区域、非法指令等
  - 系统级检查：看门狗超时、缓存访问错误、堆栈溢出、堆栈粉碎、堆栈损坏等
本文将介绍 ESP-IDF 中针对可恢复错误的错误处理机制，并提供一些常见错误的处理模式。
关于如何处理不可恢复的错误，请查阅 不可恢复错误。

### 4.8.2 错误码
ESP-IDF 中大多数函数会返回 esp_err_t 类型的错误码，esp_err_t 实质上是带符号的整型，ESP_OK 代表成功（没有错误），具体值定义为 0。
在 ESP-IDF 中，许多头文件都会使用预处理器，定义可能出现的错误代码。这些错误代码通常均以 ESP_ERR_ 前缀开头，一些常见错误（比如内存不足、超时、无效参数等）的错误代码则已经在 esp_err.h 文件中定义好了。此外，ESP-IDF 中的各种组件 (component) 也都可以针对具体情况，自行定义更多错误代码。
完整错误代码列表，请见 [错误代码参考](#) 中查看完整的错误列表。

### 4.8.3 错误码到错误消息
错误码并不直观，因此 ESP-IDF 还可以使用 esp_err_to_name() 或者 esp_err_to_name_r() 函数，将错误代码转换为具体的错误消息。例如，我们可以向 esp_err_to_name() 函数传递错误代码 0x101，可以得到返回字符串 “ESP_ERR_NO_MEM”。这样一来，我们可以在日志中输出更加直观的错误消息，而不是简单的错误码，从而帮助研发人员更快理解发生了何种错误。
此外，如果出现找不到匹配的 ESP_ERR_ 值的情况，函数 esp_err_to_name_r() 则会尝试将错误码作为一种 POSIX 错误码 进行解释。具体过程为：POSIX 错误代码（例如 ENOENT, ENOMEM）定义在 errno.h 文件中，可以通过 errno 变量获得。进而调用 strerror_r() 函数实现。在 ESP-IDF 中，errno 是一个基于线程的局部变量，即每个 FreeRTOS 任务都有自己的 errno 副本，通过函数修改 errno 也只会作用于当前任务中的 errno 变量值。
该功能（即在无法匹配 ESP_ERR_ 值时，尝试用标准 POSIX 解释错误码）默认启用。用户也可以禁用该功能，从而减小程序的二进制文件大小。详情可见 CONFIG_ESP_ERR_TO_NAME_LOOKUP 注意，该功能对禁用并不影响 esp_err_to_name() 和 esp_err_to_name_r() 函数的定义。用户仍可调用这两个函数转化错误码。在这种情况下，esp_err_to_name() 函数在遇到无法匹配错误码的情况会返回 UNKNOWN ERROR，而 esp_err_to_name_r() 函数会返回 Unknown error 0xXXXX(YYYYY)，其中 0xXXXX 和 YYYY 分别代表错误代码的十六进制和十进制表示。

### 4.8.4 ESP_ERROR_CHECK 宏
宏 ESP_ERROR_CHECK 的功能和 assert 类似，不同之处在于：这个宏会检查 esp_err_t 的值，而非判断 bool 条件。如果传给 ESP_ERROR_CHECK 的参数不等于 ESP_OK，则会在控制台上打印错误消息，然后调用 abort() 函数。
错误消息通常如下所示：

```plaintext
ESP_ERROR_CHECK failed: esp_err_t 0x107 (ESP_ERR_TIMEOUT) at 0x400d1fdf

file:  "/Users/user/esp/example/main/main.c" line 20
func: app_main
expression: sdmmc_card_init(host, &card)
Backtrace: 0x40086e7c:0x3ffbf4ff0 0x40087328:0x3ffbf5010 0x400d1fdf:0x3ffbf5030_ ←0x400d0816:0x3ffbf5050
```

**备注**：如果使用 IDF 监视器，则最后一行回溯结果中的地址将会被自动解析为相应的文件名和行号。

- 第一行打印错误代码的十六进制表示，及该错误在源代码中的标识符。这个标识符取决于 CONFIG_ESP_ERR_TO_NAME_LOOKUP 选项的设定。最后，第一行还会打印程序中该错误发生的具体位置。
Chapter 4. API

- 下面几行显示了程序中调用ESP_ERROR_CHECK宏的具体位置，以及传递给该宏的参数。
- 最后一行打印回测结果。对于所有不可恢复错误，这里在应急处理程序中打印的内容都是一样的。
  更多有关回测结果的详细信息，请参阅不可恢复错误。

### 4.8.5 ESP_ERROR_CHECK_WITHOUT_ABORT宏

宏ESP_ERROR_CHECK_WITHOUT_ABORT的功能和ESP_ERROR_CHECK类似，不同之处在于它不会调用abort()。

### 4.8.6 ESP_RETURN_ON_ERROR宏

宏ESP_RETURN_ON_ERROR用于错误码检查，如果错误码不等于ESP_OK，该宏会打印错误信息，并使原函数立刻返回。

### 4.8.7 ESP_GOTO_ON_ERROR宏

宏ESP_GOTO_ON_ERROR用于错误码检查，如果错误码不等于ESP_OK，该宏会打印错误信息，将局部变量ret赋值为该错误码，并使原函数跳转至给定的goto_tag。

### 4.8.8 ESP_RETURN_ON_FALSE宏

宏ESP_RETURN_ON_FALSE用于条件检查，如果给定条件不等于true，该宏会打印错误信息，并使原函数立刻返回，返回值为给定的err_code。

### 4.8.9 ESP_GOTO_ON_FALSE宏

宏ESP_GOTO_ON_FALSE用于条件检查，如果给定条件不等于true，该宏会打印错误信息，将局部变量ret赋值为给定的err_code，并使原函数跳转至给定的goto_tag。

### 4.8.10 CHECK宏使用示例

示例:

```c
static const char* TAG = "Test";

esp_err_t test_func(void)
{
    esp_err_t ret = ESP_OK;

    ESP_ERROR_CHECK(x); // err message_
    // printed if `x` is not `ESP_OK`, and then `abort()`.
    ESP_ERROR_CHECK_WITHOUT_ABORT(x); // err message_
    // printed if `x` is not `ESP_OK`, without `abort()`.
    ESP_RETURN_ON_ERROR(x, TAG, "fail reason 1"); // err message_
    // printed if `x` is not `ESP_OK`, and then function returns with code `x`.
    ESP_GOTO_ON_ERROR(x, err, TAG, "fail reason 2"); // err message_
    // printed if `x` is not `ESP_OK`, `ret` is set to `x`, and then jumps to `err`.
    ESP_RETURN_ON_FALSE(a, err_code, TAG, "fail reason 3"); // err message_
    // printed if `a` is not `true`, and then function returns with code `err_code`.
    ESP_GOTO_ON_FALSE(a, err_code, err, TAG, "fail reason 4"); // err message_
    // printed if `a` is not `true`, `ret` is set to `err_code`, and then jumps to `err`.

err:
```

(下页继续)
// clean up
return ret;

备份：如果 Kconfig 中的 CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT 选项被打开，CHECK 宏将不会打印错误信息，其他功能不变。

ESP_RETURN_xx 和 ESP_GOTO_xx 宏不可以在中断服务程序里被调用。如需要在中断中使用类似功能，请使用 xx_ISR 宏，如 ESP_RETURN_ON_ERROR_ISR 等。

4.8.11 错误处理模式

1. 尝试恢复。根据具体情况不同，我们具体可以:
   • 在一段时间后，重新调用该函数;
   • 尝试删除该驱动，然后重新进行“初始化”;
   • 采用其他外带机制，修改导致错误发生的条件（例如，对一直没有响应的外设进行复位等）。
   示例:

```c
esp_err_t err;
do {
   err = sdio_slave_send_queue(addr, len, arg , timeout);
   // 如果发送队列已满就不断重试
} while (err == ESP_ERR_TIMEOUT);
if (err != ESP_OK) {
   // 处理其他错误
   free(card);
   return err;
}
```

2. 将错误传回调用程序。在某些中断件组件中，采用此类处理模式代表函数必须以相同的错误码退出，这样才能确保所有分配的资源都能得到释放。
   示例:

```c
sdmmc_card_t * card = calloc(1, sizeof(sdmmc_card_t));
if (card == NULL) {
   return ESP_ERR_NO_MEM;
}

esp_err_t err = sdmmc_card_init(host, &card);
if (err != ESP_OK) {
   // 释放内存
   free(card);
   // 将错误码传给上层（例如通知用户）
   // 或者，应用程序可以自定义错误代码并返回
   return err;
}
```

3. 转为不可恢复错误，比如使用 ESP_ERROR_CHECK。详情请见 ESP_ERROR_CHECK 宏 章节。
   对于中断件组件而言，通常并不希望在发生错误时中止应用程序。不过，有时在应用程序级别，这种做法是可以接受的。
   在 ESP-IDF 的示例代码中，很多都会使用 ESP_ERROR_CHECK 来处理各种 API 引发的错误。虽然这不是应用程序的最佳做法，但可以让示例代码看起来更加简洁。
   示例:

```c
ESP_ERROR_CHECK(spi_bus_initialize(host, bus_config, dma_chan));
```

4.8.12 C++ 异常

请参考 Exception handling。
4.9 ESP-BLE-MESH

蓝牙 mesh 网络实现了无线设备的“多对多”通讯，其可用于建立包含大量设备的网络。

设备能将数据中继至不在初始设备无线电覆盖范围内的其他设备。这样，mesh 网络就能够覆盖非常大的物理区域，并且囊括大量设备。Mesh 网络非常适用于楼宇自动化，传感器网络和其他物联网解决方案，这些情景下数以十、百计，千计的设备需要与其他设备进行安全可靠的通信。

蓝牙 mesh 并非无线通信技术，而是一种网络技术。该技术基于一种无线通讯协议栈，即低功耗蓝牙。

ESP-BLE-MESH 基于 Zephyr 蓝牙 Mesh 协议栈的顶端，其实现支持设备配网和节点控制，同时也实现了代理、中继，低功耗和朋友等节点功能。

有关 ESP-BLE-MESH 架构实现的信息，请参见 ESP-BLE-MESH 架构；有关各自 API 的信息，请参见 ESP-BLE-MESH API Reference。

ESP-BLE-MESH 的实现和认证基于最新的 Mesh Profile v1.0.1。有关 ESP-BLE-MESH 认证的细节，请参考此处。

备注：如果您在寻找 ESP32 基于 Wi-Fi 的 mesh 方案，请查阅乐鑫的另一款产品 ESP-WIFI-MESH。更多相关信息及文档，请参见 ESP-WIFI-MESH。

4.9.1 ESP-BLE-MESH 快速入门

该章旨在帮助您基于乐鑫的 ESP32 开发板搭建 ESP-BLE-MESH 网络。

我们将展示如何搭建并运行一个包含 3 个节点的小型 ESP-BLE-MESH 网络，其中包含设备配网、节点配置，以及向特定节点上的 Generic OnOff Server Model 发送开关灯命令。

如果您是第一次接触 ESP-IDF，请参见 esp-idf 快速入门来设置开发环境，编译、烧写和运行示例应用程序。

硬件及软件准备

硬件:

- 3 块 ESP32 开发板，请参见 options。
- 连接开发板的 USB 线。
- ESP-IDF 开发环境。
- 运行 Android 或 iOS 的手机或平板。

软件:

- 下载至 ESP32 开发板的示例应用 bluetooth/esp_ble_mesh/ble_mesh_node。
- 手机 App: nRF Mesh Android 或 iOS 版本。除 nRF Mesh 的 App，以下 App 也支持 ESP-BLE-MESH:
  - EspBleMesh Android App
  - Silicon Labs Android 或 iOS App

安装

以下详细步骤可指导您完成安装过程。

步骤 1. 检查硬件 ESP32-DevKitC 和 ESP-WROVER-KIT 开发板均支持 ESP-BLE-MESH。您可以通过 menuconfig: idf.py menuconfig > Example Configuration > Board selection for ESP-BLE-MESH 选择特定的开发板。
备注：如果您打算使用 ESP32-DevKitC 开发板，请将 RGB 灯焊接至 GPIO 管脚 25、26 和 27。

步骤 2. 配置软件  进入 bluetooth/esp_ble_mesh/ble_mesh_node 示例文件夹，运行 idf.py menuconfig 选择所使用的开发板，然后运行 idf.py build 编译示例。

步骤 3. 下载应用  

进入 bluetooth/esp_ble_mesh/ble_mesh_node 示例编译成功后，用户可以运行 idf.py flash 将编译生成的二进制文件下载至 3 块开发板中。

当开发板上电后，RGB 灯会变为绿色。

步骤 4. 设备配网  在该章节中，我们将使用 NRF Mesh Android App 演示如何配网设备。用户也可以从 App Store 下载其 iOS 版本。

4.1 扫描 (scanner)  扫描 (Scanner) 是 NRF Mesh App 搜索蓝牙通信范围内未配网设备的功能。打开 App，点击底部的扫描按钮 Scanner。App 便会开始扫描设备，很快，我们便可在屏幕上看到 3 个未配网设备。

4.2 识别 用户可以选择任何一个未配网设备，此时 App 会尝试和该设备建立连接。连接成功（有时可能需要尝试多次），且发现相应的 ESP-BLE-MESH GATT 服务后，用户可以在屏幕上看到识别按钮 IDENTIFY。IDENTIFY 操作告诉用户哪个设备将被配网。

备注：IDENTIFY 需要设备侧的支持，然而才能用来识别当前正在配网的设备。当前如果点击识别按钮 IDENTIFY，除了串口输出的 log，在当前的 example 中设备侧不会有其他表现。
点击识别按钮 IDENTIFY 后，用户可以看到配网按钮 PROVISION。

4.3 配网 点击配网按钮 PROVISION，App 会开始配网设备。当设备配网成功后，开发板上的 RGB 灯会熄灭，此时 App 会执行以下几个步骤：

1. 和该节点（设备配网后成为节点）断开连接
2. 尝试和该节点重新建立连接
3. 连接成功并且发现了相应的 ESP-BLE-MESH GATT 服务
4. 获取节点的 Composition Data 并且给该节点添加 AppKey

当以上所有的步骤完成后，节点初始配置完成，此时点击 OK，用户可以看见节点的单播地址分配成功，并且其 Composition Data 也被成功解析。

有时在上述步骤 2 中，App 可能与节点连接失败。这种情况下，用户点击 OK 后可以看到，节点只有单播地址被成功分配，Composition data 并没有被成功获取。此时用户需要点击右上角的连接按钮 CONNECT，屏幕上会显示原先配网的节点，用户需要选择该节点并与其建立连接。

连接成功后，App 会显示获取 Composition Data 以及添加 AppKey 的按钮。

如果该设备是 App 配网的第二个或第三个节点，此时点击连接按钮 CONNECT 后，用户可以在屏幕中看到 2 个或 3 个节点。这种情况下，用户可以选择其中一个节点建立连接，连接成功后可以返回至主界面选择需要配置的节点。

这里给出了一个 3 个节点的示例。

- 左侧图片表示第三个设备成功配网，但是 App 没有和其成功建立连接。当 App 尝试去重新连接第三个节点时，界面上会出现 3 个节点。
- 右侧图片表示节点成功建立连接后，App 显示这 3 个节点的信息。用户可以看到 App 已经获取了第一个和第二个节点的 Composition Data，但是对于第三个节点，只有单播地址被成功分配而节点的 Composition Data 未知。

4.4 配置 当成功配网和初始配置完成后，用户可以配置节点的其余信息，例如将 AppKey 绑定至每个元素 (element) 的每个模型 (model) 中，设置模型的发布信息等。
图 9: nRF Mesh - 识别 - 配网

图 10: nRF Mesh - 配置完成
图 11: nRF Mesh - 初始化配置失败

图 12: nRF Mesh - 重新初始配置
下图展示了如何将 AppKey 绑定至 Primary Element 中的 Generic OnOff Server Model 上。用户不需要将 AppKey 绑定至 Configuration Server Model，因为该模型使用 DevKey 在 Upper Transport Layer 中对消息进行加密。

Step 5. 运行网络 当 3 个元素中的 Generic OnOff Server Models 均成功绑定 AppKey 后，用户可以使用 App 开关 RGB 灯。在 bluetooth/esp_ble_mesh/ble_mesh_node 示例中，第一个 Generic OnOff Server Model 用来控制 红色，第二个用来控制 绿色，同时第三个用来控制 蓝色。下图展示了打开了不同色灯的开发板。

备注：对于 nRF Mesh iOS App [version 1.0.4]，当节点包含超过一个元素时，App 表现不正确。如果用户尝试打开或者关闭第 2 个或第 3 个 Generic OnOff Server Model，App 会将相应的消息发送到第 1 个 Generic OnOff Server Model。

4.9.2 ESP-BLE-MESH 示例

- ESP-BLE-MESH 节点 - 展示了将 ESP-BLE-MESH 作为拥有 Configuration Server model 和 Generic OnOff Server model 的节点设备的用法。然后，ESP-BLE-MESH Provisioner 可以配网设备，控制表示开/关状态的 RGB 灯，示例请见 example code。
- ESP-BLE-MESH 客户端模型 - 展示了 Generic OnOff Client model 如何在节点内工作。节点拥有 Configuration Server model、Generic OnOff Server model 和 Generic OnOff Client model，示例请见 example code。
- ESP-BLE-MESH Provisioner - 展示了设备如何充当 ESP-BLE-MESH Provisioner 以配网设备。Provisioner 拥有 Configuration Server model、Configuration Client model 和 Generic OnOff Client model，示例请见 example code。
图 14: nRF Mesh - Model Bind AppKey

图 15: nRF Mesh - 通用开关控制
4.9.3 ESP-BLE-MESH 演示视频

- Espressif Fast Provisioning using ESP-BLE-MESH App
- Espressif ESP-BLE-MESH and Wi-Fi Coexistence

4.9.4 ESP-BLE-MESH 常见问题手册

1. Provisioner 开发
2. 节点开发
3. ESP-BLE-MESH 和 Wi-Fi 共存
4. 快速配网
5. Log 帮助
6. 示例帮助
7. 其他

4.9.5 相关文档
ESP-BLE-MESH Feature List

Supported Features

Mesh Core

- **Provisioning: Node Role**
  - PB-ADV and PB-GATT
  - OOB Authentication
- **Provisioning: Provisioner Role**
  - PB-ADV and PB-GATT
  - OOB Authentication
- **Networking**
  - Relay
  - Segmentation and Reassembly
  - Key Refresh Procedure
  - IV Update Procedure
  - Friend
  - Low Power
  - Proxy Server
  - Proxy Client
- **Multiple Client Models Run Simultaneously**
  - Support multiple client models send packets to different nodes simultaneously
  - No blocking between client model and server model
- **NVS Storing**
  - Store provisioning and configuration information of ESP-BLE-MESH Node

Mesh Models

- **Foundation models**
  - Configuration Server model
  - Configuration Client model
  - Health Server model
  - Health Client model
- **Generic client models**
  - Generic OnOff Client
  - Generic Level Client
  - Generic Default Transition Time Client
  - Generic Power OnOff Client
  - Generic Power Level Client
  - Generic Battery Client
  - Generic Location Client
  - Generic Property Client
- **Sensor client models**
  - Sensor Client
- **Time and Scenes client models**
  - Time Client
  - Scene Client
  - Scheduler Client
- **Lighting client models**
  - Light Lightness Client
  - Light CTL Client
  - Light HSL Client
  - Light xyL Client
  - Light LC Client
- **Generic server models**
  - Generic OnOff Server
  - Generic Level Server
– Generic Default Transition Time Server
– Generic Power OnOff Server
– Generic Power OnOff Setup Server
– Generic Power Level Server
– Generic Power Level Setup Server
– Generic Battery Server
– Generic Location Server
– Generic Location Setup Server
– Generic User Property Server
– Generic Admin Property Server
– Generic Manufacturer Property Server
– Generic Client Property Server

• Sensor server models
  – Sensor Server
  – Sensor Setup Server

• Time and Scenes server models
  – Time Server
  – Time Setup Server
  – Scene Server
  – Scene Setup Server
  – Scheduler Server
  – Scheduler Setup Server

• Lighting server models
  – Light Lightness Server
  – Light Lightness Setup Server
  – Light CTL Server
  – Light CTL Temperature Server
  – Light CTL Setup Server
  – Light HSL Server
  – Light HSL Hue Server
  – Light HSL Saturation Server
  – Light HSL Setup Server
  – Light xyl Server
  – Light xyl Setup Server
  – Light LC Server
  – Light LC Setup Server

Mesh Applications

• ESP-BLE-MESH Node
  – Tutorial
  – Tutorial
  – Example

• ESP-BLE-MESH Provisioner
  – Tutorial
  – Example

• ESP-BLE-MESH Fast Provisioning
  – Fast Provisioning Client Model Tutorial
  – Fast Provisioning Server Model Tutorial
  – Example
  – Demo Video

• ESP-BLE-MESH and Wi-Fi Coexistence
  – Tutorial
  – Example
  – Demo Video

• ESP-BLE-MESH Console Commands
  – Example
Future Release Features

Mesh Core
- Provisioner NVS Storage

Mesh Applications
- Fast OTA
- Friendship

ESP-BLE-MESH 架构

本文档将介绍 ESP-BLE-MESH 的架构概览、架构实现和辅助程序。

- ESP-BLE-MESH 架构概览
  - 描述了 ESP-BLE-MESH 架构的 5 大部分及每个部分的功能。
- ESP-BLE-MESH 架构实现
  - 描述了 ESP-BLE-MESH 文件的基本功能。文件与 ESP-BLE-MESH 架构的对应关系及文件间调用的接口。
- ESP-BLE-MESH 辅助程序
  - 描述了 ESP-BLE-MESH 的辅助程序，比如 Mesh 网络管理、Mesh 特性等。

1. ESP-BLE-MESH 架构概览 目前，ESP-BLE-MESH 已经实现了 Mesh Profile 的大多数功能及 Mesh Model 规范中定义的所有 Client Model。未支持的功能/模型尚在开发中，会尽快提供。ESP-BLE-MESH 已通过 Bluetooth SIG 蓝牙技术联盟的 认证。

ESP-BLE-MESH 架构主要由以下 5 大部分组成：

- Mesh 协议栈
  - Mesh Networking 负责 ESP-BLE-MESH 节点的网络消息处理等。
  - Mesh Provisioning 负责 ESP-BLE-MESH 设备的启动配置流程。
  - Mesh Models 负责实现 SIG 定义的模型。
- 网络管理
  - 负责实现网络管理程序，包括节点删除程序、网络索引 (IV Index) 恢复程序等。
- 特性
  - 包括 ESP-BLE-MESH 特性，如低功耗特性、好用特性、中继特性等。
- Mesh 承载层
  - 包括 广播承载层和 GATT 承载层。承载层对于 ESP-BLE-MESH 协议栈至关重要，因为协议栈基于蓝牙低功耗技术构建而成。其必须利用承载层通过 BLE 广播通道和连接通道进行数据传输。
- 应用程序
  - 基于 ESP-BLE-MESH 协议栈和 Mesh Models。
  - 通过调用 API 和处理事件，Applications 实现了与 ESP-BLE-MESH 协议栈中的 Mesh Networking 和 Mesh Provisioning 的交互，也实现了与 Mesh Models 中一系列模型的交互。

1.1 Mesh 协议栈

1.1.1 Mesh Networking 协议栈架构中的 Mesh Networking 实现了如下功能：

- Mesh 网络中节点间的通讯。
- Mesh 网络中消息的加解密。
- Mesh 网络资源的管理，如网络秘钥 (NetKey)、网络索引等。
- Mesh 网络消息的分包与重组。
图 17: 图 1.1 ESP-BLE-MESH 架构图
• 消息在不同模型间的模型映射。
• 更多功能，请参见 ESP-BLE-MESH Feature List。

Mesh Networking 功能的实现是基于层级结构的。每一层的功能如表 1.1 所示：

<table>
<thead>
<tr>
<th>层级</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>接入层</td>
<td>接入层定义应用程序数据的格式，还对上层传输层对数据包的加密和解密进行定义和控制。</td>
</tr>
<tr>
<td>上层传输层</td>
<td>上层传输层对接入层进的数据进行加解密，间接实现“传输控制消息”的特殊消息，这种消息包括了与“友谊”和心跳包相关的消息。</td>
</tr>
<tr>
<td>底层传输层</td>
<td>底层传输层处理 PDU 的分包和重组。</td>
</tr>
<tr>
<td>网络层</td>
<td>网络层定义网络消息的地址类型和格式，实现设备的中继功能。</td>
</tr>
</tbody>
</table>

1.1.2 Mesh Provisioning 协议栈架构中的 Mesh Provisioning 实现了如下功能：

• 对未配网设备的配网。
• Mesh 网络资源的分配(单播地址，网络索引和网络秘钥)。
• 配网期间对 4 种验证方法的支持。
• 更多功能，请参见 ESP-BLE-MESH Feature List。

Mesh Provisioning 功能的实现是基于层级结构的。每一层的功能如表 1.2 所示：

<table>
<thead>
<tr>
<th>层级</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provisioning PDUs</td>
<td>通过配网协议处理不同层级的 Provisioning PDUs。</td>
</tr>
<tr>
<td>Generic Provisioning PDU/Proxy PDU</td>
<td>使用 Generic Provisioning 层或代理协议层将 Provisioning PDU 传输到未配网的设备。</td>
</tr>
<tr>
<td>PB-ADV/PB-GATT</td>
<td>这些层级定义了 Provisioning PDUs 作为可分包和重组的消息进行传输的方式。</td>
</tr>
<tr>
<td>Advertising/Provisioning Service</td>
<td>Provisioning bearer 定义了会话建立的方式，该方式用来将 Generic Provisioning 层的传输包发送到设备。</td>
</tr>
</tbody>
</table>

1.1.3 Mesh Models 协议栈架构中的 Mesh Models 实现了如下功能：

• Configuration Client/Server Models
• Health Client/Server Models
• Generic Client/Server Models
• Sensor Client/Server Models
• Time and Scenes Client/Server Models
• Lighting Client/Server Models

每一层的功能如表 1.3 所示：

<table>
<thead>
<tr>
<th>层级</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>模型层</td>
<td>模型层实现用于标准化典型用户场景操作的模型，包括 Generic Client/Server Models，Sensor Client/Server Models，Time and Scenes Client/Server Models，Lighting Client/Server Models 和若干自定义模型。</td>
</tr>
<tr>
<td>基础模型层</td>
<td>基础模型层实现与 ESP-BLE-MESH 网络配置，管理和自我诊断等相关的模型。</td>
</tr>
</tbody>
</table>

1.2 Mesh 网络管理 网络管理实现了如下功能：

• 节点移除程序：用于将节点从网络中移除。
• 网络索引恢复程序：用于恢复节点的网络索引。
1.3 Mesh 特性
特性包括以下几项:
- 低功耗特性：用于降低节点的能耗。
- 好友特性：用于为低功耗节点存储消息。
- 中继特性：用于中继/转发节点通过广播承载层收到的网络 PDU。
- Proxy Server/Client 是代理协议中的两个节点角色，其使节点可以通过面向连接的承载层收发 Network PDUs、mesh beacons、代理配置消息和 Provisioning PDU。

1.4 Mesh 承载层
协议栈框架中的 承载层负责 ESP-BLE-MESH 协议栈和低功耗蓝牙核心协议间的数据传输。
承载层可视为是基于蓝牙低功耗核心协议的载体层，其实现了 ESP-BLE-MESH 协议栈数据的接收和传输。

<table>
<thead>
<tr>
<th>层级</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATT 承载层</td>
<td>GATT 承载层使用代理协议通过 GATT 连接在两个设备之间发送和接收 Proxy PDUs。</td>
</tr>
<tr>
<td>广播承载层</td>
<td>使用广播承载层时，必须使用低功耗蓝牙广播通道来发送 mesh 数据包。数据包中的 AD Type 需要设置为 mesh 数据包的类型。</td>
</tr>
</tbody>
</table>

1.5 Mesh 应用层
协议栈框架图中的 应用层通过调用 ESP-BLE-MESH 协议栈提供的 API 并处理协议栈上报的事件来实现相应地功能，有一些常见应用，比如网关、照明等。

应用层和 API / 事件之间的交互
- 应用层调用 API
  - 调用配网相关的 API 进行配网。
  - 调用模型相关的 API 发送消息。
  - 调用设备属性相关的 API 获取设备的本地信息。
- 应用层处理事件
  - 应用层的设计基于事件设计，事件将参数传送给应用层。事件主要分为两大类。
    - 调用 API 完成的事件。
      - 比如接收消息的事件。
      - 协议栈主动上报给应用层的事件。
        - 协议栈主动上报的事件。
        - 模型主动上报的事件。
  - 事件通过应用层注册的回调函数进行上报，同时回调函数中也会包含对事件的响应处理。

API / 事件与 ESP-BLE-MESH 协议栈的交互
- 用户使用的 API 主要调用 “Mesh Networking”、Mesh Provisioning 和 Mesh Models 提供的函数。
- API / 事件和协议栈的交互不会跨越协议栈的层级进行操作。比如 API 不会调用 Network Layer 相关的函数。

2. ESP-BLE-MESH 架构实现
ESP-BLE-MESH 架构的设计和实现是基于层级和模块的。具体而言，第 2.1 节 (Mesh 网络的实现)、第 2.2 节 (Mesh 配网实现) 和第 2.3 节 (Mesh 层级实现) 基于层级思想，第 2.4 节 (网络模型的实现) 基于模块思想。
- 层级思想: 基于层级思想，网络架构根据 Mesh Profile Specification 中指定的层级设计而成。每层都有独特的文件，文件包括该层的 API 等。具体设计如图 2.1 所示。
- 模块思想: 每个文件实现一个独立的功能，供其它程序调用。
Chapter 4. API

ESP-BLE-MESH 架构采用分层的方式进行设计：数据包的处理所经过的层级顺序是固定的，也就是数据包的处理过程会形成一个消息流。因此，我们可以从图 2.1 的协议栈接口图中看到消息流。

2.1 Mesh 协议栈的实现

2.1.1 Mesh Networking 实现 Mesh Networking 中的文件列表和每个文件实现的功能如表 2.1 所示：

<table>
<thead>
<tr>
<th>文件</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>access.c</td>
<td>ESP-BLE-MESH 接入层</td>
</tr>
<tr>
<td>transport.c</td>
<td>ESP-BLE-MESH 底层/上层传输层</td>
</tr>
<tr>
<td>net.c</td>
<td>ESP-BLE-MESH 网络层</td>
</tr>
<tr>
<td>adv.c</td>
<td>用于发送 ESP-BLE-MESH 广播包的任务，一个用于处理收到的广播包的回调以及用于分配 adv 缓冲区的 API</td>
</tr>
</tbody>
</table>

2.1.2 Mesh Provisioning 实现 由于 Node/Provisioner 共存的限制，Mesh Provisioning 的实现分为两大模块。实现 Node 启动配置的特定文件如表 2.2 所示：

<table>
<thead>
<tr>
<th>文件</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>prov.c</td>
<td>ESP-BLE-MESH 节点配网 (PB-ADV &amp; PB-GATT)</td>
</tr>
<tr>
<td>proxy_server.c</td>
<td>ESP-BLE-MESH 节点代理服务器相关功能</td>
</tr>
<tr>
<td>beacon.c</td>
<td>用于处理 ESP-BLE-MESH Beacon 的 API</td>
</tr>
</tbody>
</table>

实现 Provisioner 配置功能的特定文件如表 2.3 所示：

<table>
<thead>
<tr>
<th>文件</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>provisioner_prov.c</td>
<td>ESP-BLE-MESH Provisioner 配置入网 (PB-ADV &amp; PB-GATT)</td>
</tr>
<tr>
<td>proxy_client.c</td>
<td>ESP-BLE-MESH 代理客户端相关功能</td>
</tr>
<tr>
<td>provisioner_main.c</td>
<td>ESP-BLE-MESH Provisioner 网络相关功能</td>
</tr>
</tbody>
</table>

2.1.3 Mesh Models 实现 Mesh Models 用于实现节点中所包含的模型的具体功能。服务器模型主要用于维护节点状态。客户端模型主要用于获取和修改节点状态。

<table>
<thead>
<tr>
<th>文件</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfg_cli.c</td>
<td>发送 Configuration Client 消息，接收相应应答消息</td>
</tr>
<tr>
<td>cfg_srv.c</td>
<td>接收 Configuration Client 消息，发送相应应答消息</td>
</tr>
<tr>
<td>health_cli.c</td>
<td>发送 Health Client 消息，接收相应应答消息</td>
</tr>
<tr>
<td>health_srv.c</td>
<td>接收 Health Client 消息，发送适当应答消息</td>
</tr>
<tr>
<td>client_common.c</td>
<td>ESP-BLE-MESH 模型相关操作</td>
</tr>
<tr>
<td>generic_client.c</td>
<td>发送 ESP-BLE-MESH Generic Client 消息，接收相应应答消息</td>
</tr>
<tr>
<td>lighting_client.c</td>
<td>发送 ESP-BLE-MESH Lighting Client 消息，接收相应应答消息</td>
</tr>
<tr>
<td>sensor_client.c</td>
<td>发送 ESP-BLE-MESH Sensor Client 消息，接收相应应答消息</td>
</tr>
<tr>
<td>time_scene_client.c</td>
<td>发送 ESP-BLE-MESH Time Scene Client 消息，接收相应应答消息</td>
</tr>
<tr>
<td>generic_server.c</td>
<td>发送 ESP-BLE-MESH Generic Client 消息，发送相应应答消息</td>
</tr>
<tr>
<td>lighting_server.c</td>
<td>接收 ESP-BLE-MESH Lighting Client 消息，发送相应应答消息</td>
</tr>
<tr>
<td>sensor_server.c</td>
<td>接收 ESP-BLE-MESH Sensor Client 消息，发送相应应答消息</td>
</tr>
<tr>
<td>time_scene_server.c</td>
<td>接收 ESP-BLE-MESH Time Scene Client 消息，发送相应应答消息</td>
</tr>
</tbody>
</table>
2.2 Mesh Bearers 实现  Mesh Bearers 在实现时充分考虑了可移植性。当 ESP-BLE-MESH 协议栈需要移植到其它平台时，用户只需要修改 mesh_bearer_adapt.c 就能移植成功。

表 9: 表 2.5 Mesh Bearers 文件描述

<table>
<thead>
<tr>
<th>文件</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh_bearer_adapt.c</td>
<td>ESP-BLE-MESH 承载层适配文件。此文件提供用于接收和发送 ESP-BLE-MESH ADV 和 GATT 相关数据包的接口。</td>
</tr>
</tbody>
</table>

备注：mesh_bearer_adapt.c 是对 Mesh 网络框架中 Advertising Bearer 和 GATT Bearer 的实现。

2.3 Mesh Applications 实现  我们提供了一系列用于客户开发的应用示例，用户可以基于 ESP-BLE-MESH 示例 开发产品。

1. ESP-BLE-MESH 辅助程序  辅助程序指的是 ESP-BLE-MESH 协议栈中可选的功能。辅助程序的设计通常通过 CONFIG_BLE_MESH 来实现代码的裁剪。

3.1 特性  
- 低功耗
- 好友
- 中继
- 代理客户端/代理服务器

3.2 网络管理  
- 节点移除程序
- 网络索引恢复程序
- 网络索引更新程序
- 秘钥更新程序
- 网络创建程序
- NVS 存储器

3.3 辅助程序实现  采用独立模块的设计主要考虑到两个因素：
- 该模块不具备分层实现的条件，其实现可以完全独立，不需要依赖其它模块。
- 模块中的函数会被反复使用到，因此最好设计成独立模块。独立模块如表 3.1 所示：

表 10: 表 3.1 模块文件描述

<table>
<thead>
<tr>
<th>文件</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpn.c</td>
<td>ESP-BLE-MESH 低功耗功能</td>
</tr>
<tr>
<td>friend.c</td>
<td>ESP-BLE-MESH 好友功能</td>
</tr>
<tr>
<td>net.c</td>
<td>ESP-BLE-MESH 中继功能、网络创建、网络索引更新程序、网络索引恢复程序、秘钥更新程序相关功能</td>
</tr>
<tr>
<td>proxy_server.c</td>
<td>ESP-BLE-MESH 代理服务器相关功能</td>
</tr>
<tr>
<td>proxy_client.c</td>
<td>ESP-BLE-MESH 代理客户端相关功能</td>
</tr>
<tr>
<td>settings.c</td>
<td>ESP-BLE-MESH NVS 存储器功能</td>
</tr>
<tr>
<td>main.c</td>
<td>ESP-BLE-MESH 协议栈初始化，协议栈使能，节点移除相关功能</td>
</tr>
</tbody>
</table>

ESP-BLE-MESH 常见问题手册
本文汇总了 ESP-BLE-MESH 协议栈开发的常见问题及解答，全文分为 7 个章节。

1. Provisioner 开发
2. 节点开发
3. ESP-BLE-MESH 和 Wi-Fi 共存
4. 快速配网
5. Log 帮助
6. 示例帮助
7. 其他

用户可以参考这些章节，快速找到问题的答案。该文档会根据各种渠道收集的反馈进行更新。

1. Provisioner 开发 通常而言，Provisioner 用于配网未配网设备并形成 mesh 网络。组网后，设备的角色变成节点。

1.1 未配网设备加入 ESP-BLE-MESH 网络的流程是什么？
设备通过 Provisioner 加入 ESP-BLE-MESH 网络分为两个阶段，配网阶段和配置阶段。

- 部分阶段：为设备分配单播地址，添加网络密钥 (NetKey) 等。通过配网，设备加入 ESP-BLE-MESH 网络，身份从未配网设备变为节点。
- 配置阶段：为节点添加应用密钥 (AppKey)，并将应用密钥绑定到相应模块。配置期间，有些选项是可选的，比如为节点添加订阅地址、设置发布地址等。通过配置，该节点实际上可以向 Provisioner 发送消息，也可以接收来自 Provisioner 的消息。

1.2 如果 Provisioner 想要改变节点状态，其需满足什么条件？
- 需要有和节点的服务器模型相对应的客户端模型。
- 需要有节点存同的，可用于加密消息的网络密钥和应用密钥。
- 需要知道节点的地址，可以是单播地址，也可以是订阅地址。

1.3 如何使用网络密钥和应用密钥？
- 网络密钥用于加密网络层的消息。具有相同网络密钥的节点视作在同一网络中，具有不同网络密钥的节点无法进行通信。
- 应用密钥用于加密上层传输层中的消息。如果服务器模型和客户端模型绑定的应用密钥不同，则无法实现异步通信。

1.4 如何生成网络密钥或应用密钥？是否可以采用固定的网络密钥或应用密钥？
- API esp_ble_mesh_provisioner_add_local_net_key() 可以用来添加包含固定值或随机密钥的网络密钥。
- API esp_ble_mesh_provisioner_add_local_app_key() 可以用来添加包含固定值或随机密钥的应用密钥。

1.5 Provisioner 的单播地址是不是固定的？
esp_ble_mesh_prov_t 中 prov_unicast_addr 的值用于设置 Provisioner 的单播地址，只能在初始化期间设置一次，此后不能更改。

1.6 Provisioner 的地址是否可以作为节点上报状态消息的目的地址？
Provisioner 的单播地址只能在初始化期间设置一次，此后不能更改。理论而言，只要节点知道 Provisioner 的单播地址，此地址便可用作节点上报状态消息的目地地址。节点在网络配置的过程中可以知道 Provisioner 的单播地址，因为 Provisioner 在节点发送消息时，消息的源地址就是 Provisioner 的单播地址。

订阅地址也可使用。Provisioner 订阅组地址或者虚拟地址，节点向该订阅地址发送消息。
1.7 被 Provisioner 配网到 ESP-BLE-MESH 网络中的第一个节点的单播地址是不是固定的？

`esp_ble_mesh_prov_t` 中 `prov_start_address` 的值用于设置 Provisioner 配网未配网设备的起始地址，即其首先配网的节点的单播地址。单播地址只能在初始化期间设置一次，此后不能修改。

1.8 手机 App 首先配置的节点的单播地址是不是固定的？

该 App 将确定单播地址，目前大多数单播地址是固定的。

1.9 如何知道当前 Provisioner 正在配网哪个未配网设备？

`esp_ble_mesh_prov_t` 中 `prov_attention` 的值由 Provisioner 在配网过程中设置给未配网设备。该值只能在初始化期间设置一次，此后不能修改。未配网设备加入 mesh 网络后可以用特定的方式来显示自己正在配网，比如灯光闪烁，以告知 Provisioner 其正在配网。

1.10 配网过程中，认证设备共有多少种方法？提供的范例中 provided examples 使用了什么方法？

共有四种设备认证方法，即 No OOB、Static OOB、Output OOB 和 Input OOB。提供的范例使用了 No OOB 的方式。

1.11 配置入网前，未配网设备的广播包可以携带哪些信息？

- Device UUID
- OOB Info
- URL Hash (可选的)

1.12 这些信息可以用于设备识别吗？

是的。每个设备都有独一无二的 Device UUID，用户可以通过 Device UUID 识别设备。

1.13 当 Provisioner 配网的节点包含多个元素时，单播地址是如何分配的？

- Provisioner 会给设备的主元素分配一个单播地址，其余元素的单播地址在此基础上递增。
- 比如：如果一个未配网设备有三个元素，即主要元素、第二元素和第三元素。配网完成后，节点主元素的单播地址为 0x0002，节点第二元素的单播地址为 0x0003，节点第三元素的单播地址为 0x0004。

1.14 Provisioner 如何通过 Configuration Client Model 获取并且解析节点的构成数据？

- Provisioner 可以调用 `Configuration Client Model API esp_ble_mesh_config_client_set_state()` 设置参数，调用 `esp_ble_mesh_cfg_client_get_state_t` 中的 `comp_data_get` 获取节点的构成数据。
- 用户可以参考以下代码解析 Composition Data:

```c
#include <stdio.h>
#include <string.h>
#include <stdint.h>

// test date: 0C001A000100080003000001050100000080010000103103F002A00
// 0C00 1A00 0100 0800 0300 0001 05 01 0000 0080 0100 0010 0310 3F002A00

// CID is 0x000C
// PID is 0x001A
// VID is 0x0001
// CRPL is 0x0008
// Features is 0x0003 - Relay and Friend features.
// Loc is "front" - 0x0100
```

(下页继续)
// NumS is 5
// NumV is 1
// The Bluetooth SIG Models supported are: 0x0000, 0x8000, 0x0001, 0x1000, 0x1003
// The Vendor Models supported are: Company Identifier 0x003F and Model Identifier 0x002A

typedef struct {
  int16_t cid;
  int16_t pid;
  int16_t vid;
  int16_t crpl;
  int16_t features;
  int16_t all_models;
  uint8_t sig_models;
  uint8_t vnd_models;
} esp_ble_mesh_composition_head;

typedef struct {
  uint16_t model_id;
  uint16_t vendor_id;
} tsModel;

typedef struct {
  // reserve space for up to 20 SIG models
  uint16_t SIG_models[20];
  uint8_t numSIGModels;

  // reserve space for up to 4 vendor models
  tsModel Vendor_models[4];
  uint8_t numVendorModels;
} esp_ble_mesh_composition_decode;

int decode_comp_data(esp_ble_mesh_composition_head *head, esp_ble_mesh_composition_decode *data, uint8_t *mystr, int size) {
  int pos_sig_base;
  int pos_vnd_base;
  int i;

  memcpy(head, mystr, sizeof(*head));

  if (size < sizeof(*head) + head->sig_models * 2 + head->vnd_models * 4) {
    return -1;
  }

  pos_sig_base = sizeof(*head) - 1;

  for (i = 1; i < head->sig_models * 2; i++) {
    data->SIG_models[i / 2] = (mystr[i] + pos_sig_base + 1) << 8);
    printf("%d: %4.4x\n", i / 2, data->SIG_models[i / 2]);
  }

  pos_vnd_base = head->sig_models * 2 + pos_sig_base;

  for (i = 1; i < head->vnd_models * 2; i++) {
    data->Vendor_models[i / 2].model_id = (mystr[i] + pos_vnd_base + 1) << 8);
    printf("%d: %4.4x\n", i / 2, data->Vendor_models[i / 2].model_id);
  }
}
1.15 Provisioner 如何通过获取的 Composition Data 进一步配置节点？

Provisioner 通过调用 Configuration Client Model API esp_ble_mesh_config_client_set_state() 来进行如下配置。

- 正确设置参数 esp_ble_mesh_cfg_client_set_state_t 中的 app_key_add，将应用密钥添加到节点中。
- 正确设置参数 esp_ble_mesh_cfg_client_set_state_t 中的 model_sub_add，将订阅地址添加到节点的模型中。
- 正确设置参数 esp_ble_mesh_cfg_client_set_state_t 中的 model_pub_set，将发布地址添加到节点的模型中。

1.16 节点可以自己添加相应的配置吗？

本法可用于特殊情况，如测试阶段。

- 此示例展示了节点如何为自己的模型添加新的组地址。

```c
esp_err_t example_add_fast_prov_group_address(uint16_t model_id, uint16_t --group_addr)
{
    const esp_ble_mesh_comp_t *comp = NULL;
    esp_ble_mesh_elem_t *element = NULL;
    esp_ble_mesh_model_t *model = NULL;
    int i, j;
    
    if (ESP_BLE_MESH_ADDR_IS_GROUP(group_addr)) {
        return ESP_ERR_INVALID_ARG;
    }
    
    comp = esp_ble_mesh_get_composition_data();
    if (!comp) {
```

(接上页)
return ESP_FAIL;
}

for (i = 0; i < comp->element_count; i++) {
  element = comp->elements[i];
  model = esp_ble_mesh_find_sig_model(element, model_id);
  if (!model) {
    continue;
  }
  for (j = 0; j < ARRAY_SIZE(model->groups); j++) {
    if (model->groups[j] == group_addr) {
      break;
    }
  }
  if (j == ARRAY_SIZE(model->groups)) {
    ESP_LOGW(TAG, "%s: Group address already exists, element...index: %d", __func__, i);
    continue;
  }
  for (j = 0; j < ARRAY_SIZE(model->groups); j++) {
    if (model->groups[j] == ESP_BLE_MESH_ADDR_UNASSIGNED) {
      model->groups[j] = group_addr;
      break;
    }
  }
  if (j == ARRAY_SIZE(model->groups)) {
    ESP_LOGE(TAG, "%s: Model is full of group addresses, element...index: %d", __func__, i);
  }
  return ESP_OK;
}
此示例展示了如何通过 Provisioner 检测节点是否离线。

- 节点定期给 Provisioner 发送心跳包。如果 Provisioner 超过一定的时间未接收到心跳包，则视该节点离线。

备注：心跳包的设计应该采用单包（字节数小于 11 个字节）的方式，这样收发效率会更高。

1.20 Provisioner 删除网络中的节点时，需要进行哪些操作？

通常而言，Provisioner 从网络中移除节点主要涉及三个步骤：

- 首先，Provisioner 将需要移除的节点添加至“黑名单”。
- 其次，Provisioner 启动密钥更新程序。
- 最后，节点执行节点重置程序，切换自身身份为未配网设备。

1.21 在密钥更新的过程中，Provisioner 如何更新节点的网络密钥？

- 通过正确设置参数 esp_ble_mesh_cfg_client_set_state_t 中的 net_key_update，使用 Configuration Client Model API esp_ble_mesh_config_client_set_state()，Provisioner 更新节点的网络密钥。
- 通过正确设置参数 esp_ble_mesh_cfg_client_set_state_t 中的 app_key_update，使用 Configuration Client Model API esp_ble_mesh_config_client_set_state()，Provisioner 更新节点的应用密钥。

1.22 Provisioner 如何管理 mesh 网络中的节点？

ESP-BLE-MESH 在示例中实现了一些基本的节点管理功能，比如 esp_ble_mesh_store_node_info()。ESP-BLE-MESH 还提供可用于设置节点本地名称的 API esp_ble_mesh_provisioner_set_node_name() 和可用于获取节点本地名称的 API esp_ble_mesh_provisioner_get_node_name()。

1.23 Provisioner 想要控制节点的服务器模型时需要什么？

Provisioner 在控制节点的服务器模型前，必须添加相应的客户端模型。

Provisioner 应当添加本地的网络密钥和应用密钥。

- Provisioner 调用 API esp_ble_mesh_provisioner_add_local_net_key() 以添加网络密钥。
- Provisioner 调用 API esp_ble_mesh_provisioner_add_local_app_key() 以添加应用密钥。

Provisioner 应当配置自己的客户端模型。

- Provisioner 调用 API esp_ble_mesh_provisioner_bind_app_key_to_local_model() 以绑定应用密钥至自己的客户端模型。

1.24 Provisioner 如何控制节点的服务器模型？

ESP-BLE-MESH 支持所有 SIG 定义的客户端模型。Provisioner 可以使用这些客户端模型控制节点的服务器模型。客户端模型分为 6 类，每类有相应的功能。

- Configuration Client Model
  - API esp_ble_mesh_config_client_get_state() 可用于获取 Configuration Server Model 的 esp_ble_mesh_cfg_client_get_state_t 值。
  - API esp_ble_mesh_config_client_set_state() 可用于设置 Configuration Server Model 的 esp_ble_mesh_cfg_client_set_state_t 值。
- Health Client Model
Chapter 4. API

- API `esp_ble_mesh_health_client_get_state()` 可用于获取 Health Server Model 的 `esp_ble_mesh_health_client_get_state_t` 值。
- API `esp_ble_mesh_health_client_set_state()` 可用于设置 Health Server Model 的 `esp_ble_mesh_health_client_set_state_t` 值。

  • **Generic Client Models**
    - API `esp_ble_mesh_generic_client_get_state()` 可用于获取 Generic Server Model 的 `esp_ble_mesh_generic_client_get_state_t` 值。
    - API `esp_ble_mesh_generic_client_set_state()` 可用于设置 Generic Server Model 的 `esp_ble_mesh_generic_client_set_state_t` 值。

  • **Lighting Client Models**
    - API `esp_ble_mesh_light_client_get_state()` 可用于获取 Lighting Server Model 的 `esp_ble_mesh_light_client_get_state_t` 值。
    - API `esp_ble_mesh_light_client_set_state()` 可用于设置 Lighting Server Model 的 `esp_ble_mesh_light_client_set_state_t` 值。

  • **Sensor Client Models**
    - API `esp_ble_mesh_sensor_client_get_state()` 可用于获取 Sensor Server Model 的 `esp_ble_mesh_sensor_client_get_state_t` 值。
    - API `esp_ble_mesh_sensor_client_set_state()` 可用于设置 Sensor Server Model 的 `esp_ble_mesh_sensor_client_set_state_t` 值。

  • **Time and Scenes Client Models**
    - API `esp_ble_mesh_time_scene_client_get_state()` 可用于获取 Time and Scenes Server Model 的 `esp_ble_mesh_time_scene_client_get_state_t` 值。
    - API `esp_ble_mesh_time_scene_client_set_state()` 可用于设置 Time and Scenes Server Model 的 `esp_ble_mesh_time_scene_client_set_state_t` 值。

2. 节点开发

2.1 节点包含什么样的模型？

  • ESP-BLE-MESH 中，节点由一系列的模型组成，每个模型实现节点的某些功能。
  • 模型分为两种，客户端模型和服务器模型。客户端模型可以获取并设置服务器模型的状态。
  • 模型也可以分为 SIG 模型和自定义模型。SIG 模型的所有行为都由官方定义，而自定义模型的行为
    均由用户定义。

2.2 每个模型对应的消息格式是不是固定的？

  • 消息由 opcode 和 payload 组成，通过 opcode 进行区分。
  • 与模型对应的消息的类型和格式都是固定的，这意味着模型之间传输的消息是固定的。

2.3 节点的模型可以使用哪些函数发送消息？

  • 对于客户端模型，用户可以调用 API `esp_ble_mesh_client_model_send_msg()` 发送消息。
  • 对于服务器模型，用户可以调用 API `esp_ble_mesh_server_model_send_msg()` 发送消息。
  • 对于发布，用户可以调用 API `esp_ble_mesh_model_publish()` 发布消息。

2.4 如何实现消息传输不丢包？

  如果用户要实现消息传输不丢包，则需有应答的消息。等待应答的默认时间在 `CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT` 中设置。如果发送端等待应答超时，就会触发对应的超时事件。

备注：API `esp_ble_mesh_client_model_send_msg()` 中可以设置应答的超时时间。如果参数 `msg_timeout` 设为 0，那么超时时间便会采用默认值（4 秒）。
第 4 章 API

2.5 如何发送无应答的消息？
对于客户端模型，用户可以调用 API `esp_ble_mesh_client_model_send_msg()` with the parameter `need_rsp` set to `false` 发送无应答消息。

对于服务器模型，调用 API `esp_ble_mesh_server_model_send_msg()` 发送的消息总是无应答的消息。

2.6 如何为模型添加订阅地址？
通过 Configuration Client Model 添加订阅地址。

2.7 模型发送的消息和发布的消息有何不同？
调用 API `esp_ble_mesh_client_model_send_msg()` 或 `esp_ble_mesh_server_model_send_msg()` 发送的消息会在 Network Transmit 状态规定的期限内发送。

调用 API `esp_ble_mesh_model_publish()` 发布的消息将由模型发布状态决定是否发布。消息的发布一般是周期性的，或者有固定次数。发布周期和发布次数由模型发布状态控制，并且可以通过 Configuration Client Model 进行配置。

2.8 发送不分包消息时，最多可携带多少有效字节？
不分包消息的总有效载荷长度（可由用户设置）为 11 个八位组，因此，如果消息的 opcode 为 2 个八位组，则该消息可以携带 9 个八位组的有效信息。对于 vendor 消息，由于 opcode 是 3 个八位组，剩余的有效负载长度为 8 个八位组。

2.9 什么时候应该使用节点的 Relay 功能？
如果 mesh 网络中检测到的节点很稀疏，用户可以使能节点的 Relay 功能。

如果 mesh 网络中检测到的节点很密集，用户可以选择仅使能一些节点的 Relay 功能。

如果 mesh 网络大小未知，用户可以默认使能 Relay 功能。

2.10 什么时候应该使用节点的 Proxy 功能？
如果未配网设备将由电话配网，则未配网设备应该使能 Proxy 功能，因为当前几乎所有电话都不支持通过广播承载层发送 ESP-BLE-MESH 数据包，并且，未配网设备成功配网成为 Proxy 节点后，其会通过 GATT 承载层和广播承载层与 mesh 网络中的其他节点通信。

2.11 如何使用代理过滤器？
代理过滤器用于减少 Proxy Client（如手机）和 Proxy Server（如节点）之间交换的 Network PDU 的数量。另外，通过代理过滤器，Proxy Client 可以明确请求仅接收来自 Proxy Server 的某些目标地址的 mesh 消息。

2.12 Relay 节点什么时候可以中继消息？
如果要中继消息，消息需满足以下要求。

• 消息存在于 mesh 网络中。
• 消息的目的地址不是节点的单播地址。
• 消息的 TTL 值需大于 1。

2.13 如果一条消息分成几段，那么其他 Relay 节点是接收到一段消息就中继还是等接收到完整的数据包才中继？
Relay 节点收到其中一段消息时就中继，而非一直等到接收所有的消息。
2.14 使用 Low Power 功能降低功耗的原理是什么？

- 开启无线电进行收听时，设备消耗能量。使能节点的低功耗功能后，它将在大多数时间内关闭无线电功能。
- 低功耗节点和好友节点需要合作，因此低功耗节点可以以适当或较低的频率接收消息，而无需一直收听。
- 当低功耗节点有一些新消息时，好友节点将为其存储消息。低功耗节点可以间隔固定时间轮询好友节点，以查看是否有新的消息。

2.15 设备断电后上电，如何能继续在网络中进行通讯？

在 menuconfig 中启用配置 Store BLE Mesh Node configuration persistently。

2.16 如何实现将节点自检的信息发送出来？

推荐节点通过 Health Server Model 定期发布其自检结果。

2.17 节点间如何传输消息？

节点间传输信息的可能应用场景是，一旦烟雾报警检测到高浓度的烟雾，就会触发喷淋设备。有两种实现方法。
- 方法 1：喷淋设备订阅组地址。当烟雾报警器检测到高浓度的烟雾时，它会发布一条消息，该消息的目标地址是喷淋设备已订阅的组地址。
- 方法 2：Provisioner 可以配置喷淋设备的单播地址为烟雾报警器的地址。当检测到高浓度的烟雾时，烟雾报警器以喷淋设备的单播地址为目标地址，将消息发送到喷淋设备。

2.18 设备通信必须要网关吗？

- 情况 1：节点仅在 mesh 网络内通信。这种情况下，不需要网关。ESP-BLE-MESH 网络是一个泛洪的网络，网络中的消息没有固定的路径，节点与节点之间可以随意通信。
- 情况 2：如果用户想要远程控制网络，比如在到家之前打开某些节点，则需要网关。

2.19 何时使用 IV Update 更新程序？

一旦节点的底层检测到发送的消息的序列号达到临界值，IV Update 更新程序便会启用。

2.20 如何启用 IV Update 更新程序？

节点可以使用带有 Secure Network Beacon 的 IV Update 更新程序。

3. ESP-BLE-MESH 和 Wi-Fi 共存

3.1 Wi-Fi 和 ESP-BLE-MESH 共存时，支持哪些模式？

目前，只有 Wi-Fi STA 模式支持共存。

3.2 Wi-Fi 和 ESP-BLE-MESH 共存时，为什么 Wi-Fi 吞吐量很低？

未搭载 PSRAM 的 ESP32-DevKitC 开发板，Wi-Fi 和 ESP-BLE-MESH 共存可以正常运行，但是吞吐率较低。当 Wi-Fi 和 ESP-BLE-MESH 共存时，搭载 PSRAM 的 ESP32-DevKitC 速率可以在 1 Mbps 以上。

应使能 menuconfig 中的一些配置来支持 PSRAM。

- ESP32-specific --> Support for external, SPI-connected RAM --> Try to allocate memories of Wi-Fi and LWIP...
4. API

4.1 为什么需要快速配网？
通常而言，存在少量未配网设备时，用户可以逐个配置。但是如果有大量未配网设备（比如 100 个）时，逐个配置会耗费大量时间。通过快速配网，用户可以在约 50 秒内配网 100 个未配网设备。

4.2 为什么会出现 EspBleMesh App 在快速配网期间长时间等待的情况？
快速配网期间，代理节点在配置完一个节点后会断开与 APP 的连接，待所有节点配网完成后再与 APP 重新建立连接。

4.3 为什么 APP 中显示的节点地址的数量比现有的节点地址更多？
每完成一次快速配网后，开始新一次快速配网前，APP 会存有上次配网的数据，因此 APP 中显示的节点地址的数量比现有的节点地址更多。

4.4 在 EspBleMesh App 中输入的 ** count ** 值有什么用途？
此 count 值提供给 App 配置的代理节点，以决定何时提前开始 Proxy 广播信息。

4.5 运行以下示例 fast_prov_server 的节点的 Configuration Client Model 何时开始工作？
使能了 Temporary Provisioner 功能后，Configuration Client Model 会开始工作。

4.6 Temporary Provisioner 功能会一直处于使能的状态吗？
节点收到打开/关闭电灯的消息后，所有节点会禁用其 Temporary Provisioner 功能并且转化为一般节点。

5. Log 帮助 当 ESP-BLE-MESH 协议栈底层出现错误或者警告时，您可以在这儿找到这些错误和警告的含义。

5.1 ran out of retransmit attempts 代表什么？
节点发送分段消息时，由于某些原因，接收端未收到完整的消息。节点会重传消息。当重传次数达到最大重传数时，会出现该警告，当前最大重传数为 4。

5.2 Duplicate found in Network Message Cache 代表什么？
当节点收到一条消息时，它会把该消息与网络缓存中存储的消息进行比较。如果在缓存中找到相同的消息，这意味着之前已接受过该消息。则该消息会被丢弃。
5.3 Incomplete timer expired 代表什么？
当节点在一定时间段（比如 10 秒）内未收到分段消息的所有段时，则 Incomplete 计时器到时，并且出现该警告。

5.4 No matching TX context for ack 代表什么？
当节点收到一个分段 ack 且不能找到任何自己发送的与该 ack 相关的消息时，会出现该警告。

5.5 No free slots for new incoming segmented messages 代表什么？
当节点没有空间来接收新的分段消息时，会出现该警告。用户可以通过配置 CONFIG_BLE_MESH_RX_SEG_MSG_COUNT 扩大空间。

5.6 Model not bound to AppKey 0x0000 代表什么？
当节点发送带有模型的消息且该模型尚未绑定到索引为 0x0000 的应用密钥时，会出现该报警。

5.7 Busy sending message to DST xxxx 代表什么？
该错误表示节点的客户端模型已将消息发送给目标节点，并且正在等待响应。用户无法将消息发送到单播地址相同的同一节点。接收到相应的响应或计时器到时后，可以发送另一条消息。

6. 示例帮助

6.1 ESP-BLE-MESH 回调函数如何分类？
- API esp_ble_mesh_register_prov_callback() 用于注册处理配网和入网相关事件的回调函数。
- API esp_ble_mesh_register_config_client_callback() 用于注册处理 Configuration Client Model 相关事件的回调函数。
- API esp_ble_mesh_register_config_server_callback() 用于注册处理 Configuration Server Model 相关事件的回调函数。
- API esp_ble_mesh_register_health_client_callback() 用于注册处理 Health Client Model 相关事件的回调函数。
- API esp_ble_mesh_register_health_server_callback() 用于注册处理 Health Server Model 相关事件的回调函数。
- API esp_ble_mesh_register_generic_client_callback() 用于注册处理 Generic Client Models 相关事件的回调函数。
- API esp_ble_mesh_register_light_client_callback() 用于注册处理 Lighting Client Models 相关事件的回调函数。
- API esp_ble_mesh_register_sensor_client_callback() 用于注册处理 Sensor Client Model 相关事件的回调函数。
- API esp_ble_mesh_register_time_scene_client_callback() 用于注册处理 Time and Scenes Client Models 相关事件的回调函数。
- API esp_ble_mesh_register_custom_model_callback() 用于注册处理自定义模型和未实现服务器模型的相关事件的回调函数。

7. 其他

7.1 如何打印数据包？
示例使用如下函数 ESP_LOG_BUFFER_HEX() 打印信息语境，而 ESP-BLE-MESH 协议栈使用 bt_hex() 打印。
7.2 重启 ESP32 应使用哪个 API？
API `esp_restart()`。

7.3 如何监测任务栈的剩余空间？
API `vTaskList()` 可以用于定期打印任务栈的剩余空间。

7.4 如何在不更改 menuconfig 输出级别的情况下改变 log 级别？
无需使用 menuconfig，可以通过 API `esp_log_level_set()` 修改 log 的输出级别。

ESP-BLE-MESH Terminology
A device that is not a member of a mesh network is known as an unprovisioned device.

Anode is a provisioned device.

A node that supports the Relay feature and has the Relay feature enabled is known as a Relay node.

A node that supports the Proxy feature and has the Proxy feature enabled is known as a Proxy node.

A node that supports the Friend feature, has the Friend feature enabled, and has a friendship with a node that supports the Low Power feature is known as a Friend node.

A node that supports the Low Power feature and has a friendship with a node that supports the Friend feature is known as a Low Power node.

A node that is capable of adding a device to a mesh network.

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>未配网设备</td>
<td>“A device that is not a member of a mesh network is known as an unprovisioned device.”</td>
<td>示例：照明装置、温控设备、制造设备和电动门等。</td>
</tr>
<tr>
<td>节点</td>
<td>“A node is a provisioned device.”</td>
<td>经配网加入 ESP-BLE-MESH 网络后，未配网设备的那份转为节点。节点（如照明装置、温控设备、制造设备和电动门）是指能在蓝牙 ESP-BLE-MESH 网络中发送、接收或中继消息的设备，且节点可以选择性地支持一个或多个子网。</td>
</tr>
<tr>
<td>中继节点</td>
<td>“A node that supports the Relay feature and has the Relay feature enabled is known as a Relay node.”</td>
<td>中继节点接收并中继 ESP-BLE-MESH 消息，因此消息可以传输得更远。用户可以根据节点的状态来决定是否能使节点的中继功能。消息可以中继多次，每次中继为“一跳”。消息最多可有 126 跳，足以让消息在广阔的区域内传输。</td>
</tr>
<tr>
<td>代理节点</td>
<td>“A node that supports the Proxy feature and has the Proxy feature enabled is known as a Proxy node.”</td>
<td>代理节点从一个承载层（通常包括广播承载层和 GATT 承载层）接收消息，并通过另一个承载层重新发送消息。其目的是将只支持 GATT 承载层的通讯设备接入到 ESP-BLE-MESH 网络中。通常而言，手机 App 需要一个代理节点才能接入 Mesh 网络。没有代理节点，手机 App 无法与 Mesh 网络中成员通信。</td>
</tr>
<tr>
<td>好友节点</td>
<td>“A node that supports the Friend feature, has the Friend feature enabled, and has a friendship with a node that supports the Low Power feature is known as a Friend node.”</td>
<td>好友节点相当于是低功耗节点 (LPN) 的备份，可存储发往低功耗节点的消息及安全更新信息：当低功耗节点需要这些存储的信息时，这些信息便会被传输至低功耗节点。低功耗节点必须与支持好友特性的另一节点建立“友谊”，以减少其接收器的空闲。从而降低低功耗节点的功耗。低功耗节点需要找到好友节点，与好友节点建立友谊，其中涉及的过程称为“友谊建立”。低功耗节点与好友节点的加密可让低功耗节点规划对无线电的使用，从而以适当或较低的频率接收消息，无需保持收听状态。低功耗节点会轮询好友节点以查看是否有新的消息。</td>
</tr>
<tr>
<td>低功耗节点</td>
<td>“A node that supports the Low Power feature and has a friendship with a node that supports the Friend feature is known as a Low Power node.”</td>
<td>低功耗节点通过轮询从好友节点获取信息，如消息、安全更新等。</td>
</tr>
<tr>
<td>启动配置 (简称 Provisioner)</td>
<td>“A node that is capable of adding a device to a mesh network.”</td>
<td>能够配网未配网设备的设备称为启动配置设备。这一流程通常需要通过产品制造商的提供的 App 来实现，并可在网关、智能手机、平板电脑和其他载体上使用。</td>
</tr>
</tbody>
</table>
表 12: 表 2 ESP-BLE-MESH 术语 - 节点构成

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>状态</td>
<td>“A value representing a condition of an element that is exposed by an element of a node.”</td>
<td>ESP-BLE-MESH 网络中的每台设备都具有一组独立的状态值。表示设备的某些状态，比如照明设备的亮度、颜色等状态。更改状态值会修改设备本身的物理状态，比如更改设备的开关状态值实际是在打开或关闭设备。</td>
</tr>
<tr>
<td>模型</td>
<td>“A model defines the basic functionality of a node.”</td>
<td>一个节点可能包含多个模型，而每个模型定义了节点的基本功能，比如节点所需要的状态、控制状态的消息以及处理消息所产生的动作等。节点功能的实现是基于模型的，模型可分为 SIG 模型和自定义模型，前者由 SIG 定义，而后者由用户定义。</td>
</tr>
<tr>
<td>元素</td>
<td>“An addressable entity within a device.”</td>
<td>一个节点可以包含一个或多个元素，每个元素都有一个单播地址和一个或多个模型，并且同一元素所包含的模型不可以出现重复。</td>
</tr>
<tr>
<td>节点构成状态</td>
<td>“The Composition Data state contains information about a node, the elements it includes, and the supported models.”</td>
<td>通过读取节点构成状态的值，用户可以了解节点的基本信息，比如元素的数量及每个元素中的模型。 Provisioner 通过获取这个消息对设备进一步配置，比如配置节点的订阅地址与发布地址。</td>
</tr>
</tbody>
</table>

表 13: 表 3 ESP-BLE-MESH 术语 - 特性

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>低功耗特性</td>
<td>“The ability to operate within a mesh network at significantly reduced receiver duty cycles only in conjunction with a node supporting the Friend feature.”</td>
<td>低功耗功能可降低节点的功耗。当低功耗节点寻找好友节点，且附近有多个好友节点时，它会通过算法选择最适合的好友节点。</td>
</tr>
<tr>
<td>好友特性</td>
<td>“The ability to help a node supporting the Low Power feature to operate by storing messages destined for those nodes.”</td>
<td>通过使能好友特性，节点可以使用存储低功耗节点的信息。使能了好友特性的节点可能会产生更大的功耗和内存消耗。</td>
</tr>
<tr>
<td>中继特性</td>
<td>“The ability to receive and retransmit mesh messages over the advertising bearer to enable larger networks.”</td>
<td>中继特性能让 ESP-BLE-MESH 的消息在网络之间实现多次跳跃，传输距离可超过两个节点之间直接进行无线电传输的范围，从而覆盖整个网络。中继节点中继消息时，只中继其所在子网的消息，不中继其它子网的消息。使能了中继特性的节点中继分段消息时不会考虑数据的完整性。节点每收到一条分段消息便直接中继，不会等待收到完整的消息。</td>
</tr>
<tr>
<td>代理特性</td>
<td>“The ability to receive and retransmit mesh messages between GATT and advertising bearers.”</td>
<td>代理特性的目的是允许不具备广播承载层的节点访问 ESP-BLE-MESH 网络。代理特性通常为需要和手机 App 连接的节点所用。</td>
</tr>
</tbody>
</table>
表 14: 表 4 ESP-BLE-MESH 术语 - 配置入网

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>PB-ADV</td>
<td>“PB-ADV is a provisioning bearer used to provision a device using Generic Provisioning PDUs over the advertising channels.”</td>
<td>PB-ADV 通过广播通道传输配网过程中产生的数据包，只有 Provisioner 和未配网设备都支持 PB-ADV 时才可使用这种方法进行配网。</td>
</tr>
<tr>
<td>PB-GATT</td>
<td>“PB-GATT is a provisioning bearer used to provision a device using Proxy PDUs to encapsulate Provisioning PDUs within the Mesh Provisioning Service.”</td>
<td>PB-GATT 通过连接通道传输配网过程中产生的数据包，如果未配网设备想使用此方式进行配网，其需实现相关的 Mesh Provisioning Service。未实现此服务的未配网设备不能通过 PB-GATT 承载层配网接入 mesh 网络。</td>
</tr>
<tr>
<td>配置入网</td>
<td>“Provisioning is a process of adding an unprovisioned device to a mesh network, managed by a Provisioner.”</td>
<td>经过配网，“未配网设备” 的身份转变为“节点”，成为 ESP-BLE-MESH 网络中的一员。</td>
</tr>
<tr>
<td>认证方式</td>
<td>“Authentication is a step during the provisioning of nodes.”</td>
<td>未配网设备有四种认证方法：输入带外 (Input OOB)、输出带外 (Output OOB)、静态带外 (Static OOB) 和无带外 (No OOB)。</td>
</tr>
<tr>
<td>输入带外</td>
<td>Input Out-of-Band</td>
<td>比如,  Provisioner 生成并显示随机数字，然后提示用户采取适当操作将随机数输入未配网的设备中。以照明开关为例，用户可以在一定时间内数次按下按钮，以这种形式输入 Provisioner 显示的随机数。输入带外认证方法与输出带外的认证方法类似，但设备的角色相反。</td>
</tr>
<tr>
<td>输出带外</td>
<td>Output Out-of-Band</td>
<td>比如, 未配网设备会选择一个随机数,并通过与其功能兼容的方式输出该数字。如果未配网设备是一个灯泡，则其能够闪烁指定的次数。如果未配网设备有 LCD 屏幕，则可以将随机数显示为多位数值。启动 Provisioner 的用户需要输入到显示的数字，来认证未配网的设备。</td>
</tr>
<tr>
<td>静态带外</td>
<td>Static Out-of-Band</td>
<td>静态 OOB 的认证方法：使用静态 OOB 信息。如果需要使用无 OOB 信息，请将静态 OOB 字段赋值为 0。如果需要使用 OOB 信息，请使用静态 OOB 信息认证正在配网的设备。</td>
</tr>
<tr>
<td>无带外</td>
<td>No Out-of-Band</td>
<td>无 OOB 的认证方法：将“静态 OOB” 字段赋值为 0，采用这种方式相当于不认证未配网的设备。</td>
</tr>
</tbody>
</table>
表 15: 表 5 ESP-BLE-MESH 术语 - 地址

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>未分配地址</td>
<td>“This is a special address type, with a value of 0x0000. Its use indicates that an Element has not yet been configured or had a Unicast Address assigned to it.”</td>
<td>未分配的元素地址或未分配的元素地址都称为未分配地址。鉴于这些元素没有固定的地址，它们不会用于消息的传输。建议在设置用户代码的地址之前，将该地址的值设为未分配地址。</td>
</tr>
<tr>
<td>单播地址</td>
<td>“A unicast address is a unique address allocated to each element.”</td>
<td>在配网期间，Provisioner 会向网络中处于生命周期内节点的每个元素分配一个单播地址。单播地址可能会出现在消息的源/目标地址字段中。发送到单播地址的消息只能由拥有该单播地址的元素进行处理。</td>
</tr>
<tr>
<td>虚拟地址</td>
<td>“A virtual address represents a set of destination addresses. Each virtual address logically represents a Label UUID, which is a 128-bit value that does not have to be managed centrally.”</td>
<td>虚拟地址与特定的 UUID 标签相关联，可以用作模型的发布地址或订阅地址。UUID 标签是与一个或多个节点成员相关的 128 位值。虚拟地址的第 15 位和第 14 位分别设置为 1 和 0，从第 13 位到第 0 位设置为散列值（提供 16384 个散列值）。散列是 UUID 标签的派生。使用订阅元器件检查完整的 128 位 UUID 十分低效，而散列值提供了一种更有效的方法来确定最终将哪些消息发送到哪些元器件。</td>
</tr>
<tr>
<td>群组地址</td>
<td>“A group address is an address that is programmed into zero or more elements.”</td>
<td>群组地址是 ESP-BLE-MESH 网络中的另一种多播地址。通常用于将节点进行分组。发送到 all-proxies 地址的消息应有启用了代理功能的所有节点的主要元器件处理。发送到 all-friends 地址的消息应有启用了好友功能的所有节点的主要元器件处理。发送到 all-relays 地址的消息应有启用了中继功能的所有节点的主要元器件处理。发送到 all-nodes 地址的消息应有所有节点的主要元器件处理。</td>
</tr>
</tbody>
</table>

表 16: 表 6 ESP-BLE-MESH 术语 - 安全

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>设备密钥 (DevKey)</td>
<td>“There is also a device key, which is a special application key that is unique to each node, is known only to the node and a Configuration Client, and is used to secure communications between the node and a Configuration Client.”</td>
<td>设备密钥能够使网状网络设备、配置节点。设备密钥用来加密配置信息，即配置设备时 Provisioner 和节点之间传输的消息。</td>
</tr>
<tr>
<td>应用密钥 (App-Key)</td>
<td>“Application keys are used to secure communications at the upper transport layer.”</td>
<td>应用密钥用于应用数据通信至应用层过程中对应用数据的解密，和应用层下发过程中对数据的加密。网络中的这些节点由特定的用途，并且可以根据应用程序的需求对一些敏感数据的访问进行限制。通过特定的应用密钥，这些节点有特定的密钥文件，包含密钥（用户密钥）和密钥（应用密钥）。应用密钥绑定在网络密钥上，这意味着应用密钥仅在绑定网络密钥的情况下使用。每一个应用密钥仅可绑定到一个网络密钥。</td>
</tr>
<tr>
<td>主安全资料</td>
<td>“The master security material is derived from the network key (NetKey) and can be used by other nodes in the same network. Messages encrypted with master security material can be decoded by any node in the same network.”</td>
<td>使用好友安全文件加密的相应信息消息：1. 好友查询 (Friend Poll)，2. 好友更新 (Friend Update)，3. 好友订阅列表 (Friend Subscription List)。将添加/删除/确认/友好节点发送到低功耗节点的“已存储消息”使用主安全材料加密的相应信息消息：1. 好友清除 (Friend Clear)，2. 好友清除确认 (Friend Clear Confirm)。根据应用程序的设置，低功耗节点发送到好友节点的消息会使用好友安全材料进行加密，而低功耗节点与好友节点之间的消息传输，则会使用其他网络消息。</td>
</tr>
</tbody>
</table>
### Table 17: ESP-BLE-MESH Terms - Message

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>重组/分包</td>
<td>“Segmentation and reassembly (SAR) is a method of communication network, which is divided into small units before transmitting packets and re-assembled in a proper order at the communication receiving end.”</td>
<td>底层传输层会自动分包过大的消息。接收端会回复一条应答消息，根据应答消息，发送端会重新向接收端发送其未接收到的数据包。这些都是底层传输层自动完成的。未分包的消息最多携带 15 个字节，其中 4 个字节是 transMIC，所以剩余 11 个字节；在分包的情况下，每个包每包有 12 个有效字节，最后一个包中有 8 个有效字节。特殊情况下：一个较短的包需要底层传输端强制分包，这种情况下有 8 个有效字节。</td>
</tr>
</tbody>
</table>

| 无应答/有应答 | “There are two types of messages: Un-acknowledged or Acknowledged.” | 根据接收端是否需要发送应答消息，发送的消息可分为两种。发送端需要设置最大重传次数。 |

### Table 18: ESP-BLE-MESH Terms - Basemodel

<table>
<thead>
<tr>
<th>术语</th>
<th>官方定义</th>
<th>详细说明</th>
</tr>
</thead>
</table>
| Configuration Server Model | “This model is used to represent a mesh network configuration of a device.” | 节点必须包含 Configuration Server Model。其负责维护配置相关的状态。Configuration Server Model 维护的状态包含：网络密钥名单 (NetKey List)、应用密钥名单 (AppKey List)、模型绑定的应用密钥名单 (Model to App-Key List)、节点身份 (Node Identity)、密钥刷新阶段 (Key Refresh Phase)、心跳消息发布 (Heartbeat Publish)、心跳消息订阅 (Heartbeat Subscription)、网络传输 (Network Transmit) 和中继转发 (Relay Retransmit) 等。

| Configuration Client Model | “The model is used to represent an element that can control and monitor the configuration of a node.” | Configuration Client Model 通过消息控制 Configuration Server Model 的状态。Provisioner 必须包含 Configuration Client Model。有了该模型才可发送 “Configuration Composition Data Get” 等配置消息。

| Health Server Model | “This model is used to represent a mesh network diagnostics of a device.” | Health Server Model 主要用于设备检查自身状态，查看自身是否发生错误。Health Server model 维护的状态包含：当前故障 (Current Fault)、已登记故障 (Registered Fault)、健康周期 (Health Period) 和关注时器 (Attention Timer)。 |

| Health Client Model | “The model is used to represent an element that can control and monitor the health of a node.” | Health Client Model 通过消息控制 Health Server Model 维护的状态。该模型可通过消息 “Health Fault Get” 获取其他节点的自检信息。 |
This procedure is used when the security of one or more network keys and/or one or more of the application keys has been compromised or could be compromised.

“IV 更新程序用于更新 ESP-BLE-MESH 网络的 IV Index 的值。这个值和消息加密时所需的随机数相关。为了保证随机数的值不重复，所以将这个值定期增加。IV Index 是一个 32 位的值，是一种共享网络资源。比如一个 mesh 网中的所有节点共享一个 IV Index 值。IV Index 从 0x00000000 开始，在 IV 更新过程中递增，并由特定的进程维护。以保证整个 Mesh 网内共享一个 IV Index。当节点认为它有耗尽其序列号的风险，或它确定另一个节点即将耗尽其序列号时，可以启动该程序。注意：每次的更新时间不得低于 96 小时。节点接收到 secure network beacon 或者确定自己的序列号大于特定值时，会触发 IV 更新程序。”

ESP-IDF FreeRTOS (SMP)

This document assumes that the reader has a requisite understanding of Vanilla FreeRTOS (its features, behavior, and API usage). Refer to the Vanilla FreeRTOS documentation for more details.

This document describes the API and behavioral differences between Vanilla FreeRTOS and ESP-IDF FreeRTOS that were made in order to support Symmetric Multiprocessing (SMP). This document is split into the following parts.

Contents

- ESP-IDF FreeRTOS (SMP)
  - Overview
  - Symmetric Multiprocessing
    - Basic Concepts
    - SMP on an ESP Target
  - Tasks
    - Creation
    - Execution
    - Deletion
4.10.1 Overview

The original FreeRTOS (hereinafter referred to as Vanilla FreeRTOS) is a small and efficient Real Time Operating System supported on many single-core MCUs and SoCs. However, numerous ESP targets (such as the ESP32 and ESP32-S3) are capable of dual core symmetric multiprocessing (SMP). Therefore, the version of FreeRTOS used in ESP-IDF (hereinafter referred to as ESP-IDF FreeRTOS) is a modified version of Vanilla FreeRTOS v10.4.3. These modifications allow ESP-IDF FreeRTOS to utilize the dual core SMP capabilities of ESP SoCs.

For information regarding features that have been added to ESP-IDF FreeRTOS, see ESP-IDF FreeRTOS Additions.

For a detailed ESP-IDF FreeRTOS API Reference, see FreeRTOS API reference.

4.10.2 Symmetric Multiprocessing

Basic Concepts

SMP (Symmetric Multiprocessing) is a computing architecture where two or more identical CPUs (cores) are connected to a single shared main memory and controlled by a single operating system. In general, an SMP system ...

- has multiple cores running independently. Each core has its own register file, interrupts, and interrupt handling.
- presents an identical view of memory to each core. Thus a piece of code that accesses a particular memory address will have the same effect regardless of which core it runs on.

The main advantages of an SMP system compared to single core or Asymmetric Multiprocessing systems are that...

- the presence of multiple CPUs allows for multiple hardware threads, thus increases overall processing throughput.
- having symmetric memory means that threads can switch cores during execution. This in general can lead to better CPU utilization.

Although an SMP system allows threads to switch cores, there are scenarios where a thread must/should only run on a particular core. Therefore, threads in an SMP systems will also have a core affinity that specifies which particular core the thread is allowed to run on.

- A thread that is pinned to a particular core will only be able to run on that core
- A thread that is unpinned will be allowed to switch between cores during execution instead of being pinned to a particular core.
SMP on an ESP Target

ESP targets (such as the ESP32, ESP32-S3) are dual core SMP SoCs. These targets have the following hardware features that make them SMP capable:

- Two identical cores known as CPU0 (i.e., Protocol CPU or PRO_CPU) and CPU1 (i.e., Application CPU or APP_CPU). This means that the execution of a piece of code is identical regardless of which core it runs on.
- Symmetric memory (with some small exceptions).
  - If multiple cores access the same memory address, their access will be serialized at the memory bus level.
  - True atomic access to the same memory address is achieved via an atomic compare-and-swap instruction provided by the ISA.
- Cross-core interrupts that allow one CPU to trigger and interrupt on another CPU. This allows cores to signal each other.

注: The “PRO_CPU” and “APP_CPU” aliases for CPU0 and CPU1 exist in ESP-IDF as they reflect how typical IDF applications will utilize the two CPUs. Typically, the tasks responsible for handling wireless networking (e.g., WiFi or Bluetooth) will be pinned to CPU0 (thus the name PRO_CPU), whereas the tasks handling the remainder of the application will be pinned to CPU1 (thus the name APP_CPU).

4.10.3 Tasks

Creation

Vanilla FreeRTOS provides the following functions to create a task:

- `xTaskCreate()` creates a task. The task’s memory is dynamically allocated.
- `xTaskCreateStatic()` creates a task. The task’s memory is statically allocated (i.e., provided by the user).

However, in an SMP system, tasks need to be assigned a particular affinity. Therefore, ESP-IDF provides a `PinnedToCore` version of Vanilla FreeRTOS’s task creation functions:

- `xTaskCreatePinnedToCore()` creates a task with a particular core affinity. The task’s memory is dynamically allocated.
- `xTaskCreateStaticPinnedToCore()` creates a task with a particular core affinity. The task’s memory is statically allocated (i.e., provided by the user).

The `PinnedToCore` versions of the task creation functions API differ from their vanilla counterparts by having an extra `xCOREID` parameter that is used to specify the created task’s core affinity. The valid values for core affinity are:

- 0 which pins the created task to CPU0
- 1 which pins the created task to CPU1
- `tskNO_AFFINITY` which allows the task to be run on both CPUs

Note that ESP-IDF FreeRTOS still supports the vanilla versions of the task creation functions. However, they have been modified to simply call their `PinnedToCore` counterparts with `tskNO_AFFINITY`.

备注：ESP-IDF FreeRTOS also changes the units of `ulStackDepth` in the task creation functions. Task stack sizes in Vanilla FreeRTOS are specified in number of words, whereas in ESP-IDF FreeRTOS, the task stack sizes are specified in bytes.

Execution

The anatomy of a task in ESP-IDF FreeRTOS is the same as Vanilla FreeRTOS. More specifically, ESP-IDF FreeRTOS tasks:
• Can only be in one of following states: Running, Ready, Blocked, or Suspended.
• Task functions are typically implemented as an infinite loop
• Task functions should never return

Deletion

Task deletion in Vanilla FreeRTOS is called via `vTaskDelete()`. The function allows deletion of another task or the currently running task (if the provided task handle is `NULL`). The actual freeing of the task’s memory is sometimes delegated to the idle task (if the task being deleted is the currently running task).

ESP-IDF FreeRTOS provides the same `vTaskDelete()` function. However, due to the dual core nature, there are some behavioral differences when calling `vTaskDelete()` in ESP-IDF FreeRTOS:

• When deleting a task that is pinned to the other core, that task’s memory is always freed by the idle task of the other core (due to the need to clear FPU registers).
• When deleting a task that is currently running on the other core, a yield is triggered on the other core and the task’s memory is freed by one of the idle tasks (depending on the task’s core affinity)
• A deleted task’s memory is freed immediately if:
  – The tasks is currently running on this core and is also pinned to this core
  – The task is not currently running and is not pinned to any core

Users should avoid calling `vTaskDelete()` on a task that is currently running on the other core. This is due to the fact that it is difficult to know what the task currently running on the other core is executing, thus can lead to unpredictable behavior such as:

• Deleting a task that is holding a mutex
• Deleting a task that has yet to free memory it previously allocated

Where possible, users should design their application such that `vTaskDelete()` is only ever called on tasks in a known state. For example:

• Tasks self deleting (via `vTaskDelete(NULL)`) when their execution is complete and have also cleaned up all resources used within the task.
• Tasks placing themselves in the suspend state (via `vTaskSuspend()`) before being deleted by another task.

4.10.4 SMP Scheduler

The Vanilla FreeRTOS scheduler is best described as a Fixed Priority Preemptive scheduler with Time Slicing meaning that:

• Each tasks is given a constant priority upon creation. The scheduler executes highest priority ready state task
• The scheduler can switch execution to another task without the cooperation of the currently running task
• The scheduler will periodically switch execution between ready state tasks of the same priority (in a round robin fashion). Time slicing is governed by a tick interrupt.

The ESP-IDF FreeRTOS scheduler supports the same scheduling features (i.e., Fixed Priority, Preemption, and Time Slicing) albeit with some small behavioral differences.

Fixed Priority

In Vanilla FreeRTOS, when scheduler selects a new task to run, it will always select the current highest priority ready state task. In ESP-IDF FreeRTOS, each core will independently schedule tasks to run. When a particular core selects a task, the core will select the highest priority ready state task that can be run by the core. A task can be run by the core if:

• The task has a compatible affinity (i.e., is either pinned to that core or is unpinned)
• The task is not currently being run by another core

However, users should not assume that the two highest priority ready state tasks are always run by the scheduler as a task’s core affinity must also be accounted for. For example, given the following tasks:
Chapter 4. API

- Task A of priority 10 pinned to CPU0
- Task B of priority 9 pinned to CPU0
- Task C of priority 8 pinned to CPU1

The resulting schedule will have Task A running on CPU0 and Task C running on CPU1. Task B is not run even though it is the second highest priority task.

Preemption

In Vanilla FreeRTOS, the scheduler can preempt the currently running task if a higher priority task becomes ready to execute. Likewise in ESP-IDF FreeRTOS, each core can be individually preempted by the scheduler if the scheduler determines that a higher priority task can run on that core.

However, there are some instances where a higher priority task that becomes ready can be run on multiple cores. In this case, the scheduler will only preempt one core. The scheduler always gives preference to the current core when multiple cores can be preempted. In other words, if the higher priority ready task is unpinned and has a higher priority than the current priority of both cores, the scheduler will always choose to preempt the current core. For example, given the following tasks:

- Task A of priority 8 currently running on CPU0
- Task B of priority 9 currently running on CPU1
- Task C of priority 10 that is unpinned and was unblocked by Task B

The resulting schedule will have Task A running on CPU0 and Task C preempting Task B given that the scheduler always gives preference to the current core.

Time Slicing

The Vanilla FreeRTOS scheduler implements time slicing meaning that if current highest ready priority contains multiple ready tasks, the scheduler will switch between those tasks periodically in a round robin fashion.

However, in ESP-IDF FreeRTOS, it is not possible to implement perfect Round Robin time slicing due to the following reasons:

- The task is pinned to the another core.
- For unpinned tasks, the task is already being run by another core.

Therefore, when a core searches the ready state task list for a task to run, the core may need to skip over a few tasks in the same priority list or drop to a lower priority in order to find a ready state task that the core can run.

The ESP-IDF FreeRTOS scheduler implements a Best Effort Round Robin time slicing for ready state tasks of the same priority by ensuring that tasks that have been selected to run will be placed at the back of the list, thus giving unselected tasks a higher priority on the next scheduling iteration (i.e., the next tick interrupt or yield).

The following example demonstrates the Best Effort Round Robin time slicing in action. Assume that:

- There are four ready state tasks of the same priority AX, B0, C1, D1 where: - The priority is the current highest priority with ready state tasks - The first character represents the task’s names (i.e., A, B, C, D) - And the second character represents the tasks core pinning (and X means unpinned)
- The task list is always searched from the head

| Starting state. None of the ready state tasks have been selected to run |
| Head [ AX , B0 , C1 , D0 ] Tail |
| Core 0 has tick interrupt and searches for a task to run. |
| Task A is selected and is moved to the back of the list |
Chapter 4. API

Tick Interrupts

Vanilla FreeRTOS requires that a periodic tick interrupt occurs. The tick interrupt is responsible for:

- Incrementing the scheduler’s tick count
- Unblocking any blocked tasks that have timed out
- Checking if time slicing is required (i.e., triggering a context switch)
- Executing the application tick hook

Tick Interrupts

Vanilla FreeRTOS requires that a periodic tick interrupt occurs. The tick interrupt is responsible for:

- Incrementing the scheduler’s tick count
- Unblocking any blocked tasks that have timed out
- Checking if time slicing is required (i.e., triggering a context switch)
- Executing the application tick hook

Tick Interrupts

Vanilla FreeRTOS requires that a periodic tick interrupt occurs. The tick interrupt is responsible for:

- Incrementing the scheduler’s tick count
- Unblocking any blocked tasks that have timed out
- Checking if time slicing is required (i.e., triggering a context switch)
- Executing the application tick hook

Tick Interrupts

Vanilla FreeRTOS requires that a periodic tick interrupt occurs. The tick interrupt is responsible for:

- Incrementing the scheduler’s tick count
- Unblocking any blocked tasks that have timed out
- Checking if time slicing is required (i.e., triggering a context switch)
- Executing the application tick hook
In ESP-IDF FreeRTOS, each core will receive a periodic interrupt and independently run the tick interrupt. The tick interrupts on each core are of the same period but can be out of phase. However, the tick responsibilities listed above are not run by all cores:

- CPU0 will execute all of the tick interrupt responsibilities listed above
- CPU1 will only check for time slicing and execute the application tick hook

In ESP-IDF FreeRTOS, a separate pinned idle task is created for each core. The idle tasks on each core have the same responsibilities as their vanilla counterparts.

Scheduler Suspension

Vanilla FreeRTOS allows the scheduler to be suspended/resumed by calling `vTaskSuspendAll()` and `xTaskResumeAll()` respectively. While the scheduler is suspended:

- Task switching is disabled but interrupts are left enabled.
- Calling any blocking/yielding function is forbidden, and time slicing is disabled.
- The tick count is frozen (but the tick interrupt will still occur to execute the application tick hook)

On scheduler resumption, `xTaskResumeAll()` will catch up all of the lost ticks and unblock any timed out tasks.

In ESP-IDF FreeRTOS, suspending the scheduler across multiple cores is not possible. Therefore when `vTaskSuspendAll()` is called on a particular core (e.g., core A):

- Task switching is disabled only on core A but interrupts for core A are left enabled.
- Calling any blocking/yielding function on core A is forbidden. Time slicing is disabled on core A.
- If an interrupt on core A unblocks any tasks, those tasks will go into core A’s own pending ready task list
- If core A is CPU0, the tick count is frozen and a pended tick count is incremented instead. However, the tick interrupt will still occur in order to execute the application tick hook.

When `xTaskResumeAll()` is called on a particular core (e.g., core A):

- Any tasks added to core A’s pending ready task list will be resumed.
- If core A is CPU0, the pended tick count is unwound to catch up the lost ticks.

Disabling Interrupts

Vanilla FreeRTOS allows interrupts to be disabled and enabled by calling `taskDISABLE_INTERRUPTS` and `taskENABLE_INTERRUPTS` respectively.

ESP-IDF FreeRTOS provides the same API, however interrupts will only disabled or enabled on the current core.
### 4. API

#### 7. Disabling interrupts is a valid method of achieve mutual exclusion in Vanilla FreeRTOS (and single core systems in general). However, in an SMP system, disabling interrupts is **NOT** a valid method ensuring mutual exclusion. Refer to Critical Sections for more details.

#### 8. Startup and Termination

ESP-IDF FreeRTOS does **not** require users to call `vTaskStartScheduler()` to start the scheduler. The startup flow of an ESP-IDF application will already call this automatically. The entry point for user code is a user defined `void app_main(void)` function. For more details regarding the startup of ESP-IDF FreeRTOS applications, see ESP-IDF FreeRTOS Applications.

ESP-IDF FreeRTOS does **not** support scheduler termination. Calling `vTaskEndScheduler()` will simply cause the application to abort.

#### 4.10.5 Critical Sections

##### API Changes

Vanilla FreeRTOS implements critical sections by disabling interrupts. This prevents preemptive context switches and the servicing of ISRs during a critical section. Thus a task/ISR that enters a critical section is guaranteed to be the sole entity to access a shared resource. Critical sections in Vanilla FreeRTOS have the following API:

- `taskENTER_CRITICAL()` enters a critical section by disabling interrupts
- `taskEXIT_CRITICAL()` exits a critical section by reenabling interrupts
- `taskENTER_CRITICAL_FROM_ISR()` enters a critical section from an ISR by disabling interrupt nesting
- `taskEXIT_CRITICAL_FROM_ISR()` exits a critical section from an ISR by reenabling interrupt nesting

However, in an SMP system, merely disabling interrupts does not constitute a critical section as the presence of other cores means that a shared resource can still be concurrently accessed. Therefore, critical sections in ESP-IDF FreeRTOS are implemented using spinlocks. To accommodate the spinlocks, the ESP-IDF FreeRTOS critical section APIs contain an additional spinlock parameter as shown below:

- Spinlocks are of `portMUX_TYPE` (**not to be confused to FreeRTOS mutexes**)
- `taskENTER_CRITICAL(&mux)` enters a critical from a task context
- `taskEXIT_CRITICAL(&mux)` exits a critical section from a task context
- `taskENTER_CRITICAL_ISR(&mux)` enters a critical section from an interrupt context
- `taskEXIT_CRITICAL_ISR(&mux)` exits a critical section from an interrupt context

**Note:** The critical section API can be called recursively (i.e., nested critical sections). Entering a critical section multiple times recursively is valid so long as the critical section is exited the same number of times it was entered. However, given that critical sections can target different spinlocks, users should take care to avoid dead locking when entering critical sections recursively.

##### Implementation

In ESP-IDF FreeRTOS, the process of a particular core entering and exiting a critical section is as follows:

- For `taskENTER_CRITICAL(&mux)` (or `taskENTER_CRITICAL_ISR(&mux)`)
  1. The core disables its interrupts (or interrupt nesting) up to `configMAX_SYSCALL_INTERRUPT_PRIORITY`
  2. The core then spins on the spinlock using an atomic compare-and-set instruction until it acquires the lock. A lock is acquired when the core is able to set the lock’s owner value to the core’s ID.
  3. Once the spinlock is acquired, the function returns. The remainder of the critical section runs with interrupts (or interrupt nesting) disabled.
- For `taskEXIT_CRITICAL(&mux)` (or `taskEXIT_CRITICAL_ISR(&mux)`)

---

Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151

Espressif Systems 2207
1. The core releases the spinlock by clearing the spinlock’ s owner value
2. The core re-enables interrupts (or interrupt nesting)

Restrictions and Considerations

Given that interrupts (or interrupt nesting) are disabled during a critical section, there are multiple restrictions regarding what can be done within a critical sections. During a critical section, users should keep the following restrictions and considerations in mind:

- Critical sections should be as kept as short as possible
  - The longer the critical section lasts, the longer a pending interrupt can be delayed.
  - A typical critical section should only access a few data structures and/or hardware registers
  - If possible, defer as much processing and/or event handling to the outside of critical sections.
- FreeRTOS API should not be called from within a critical section
- Users should never call any blocking or yielding functions within a critical section

4.10.6 Misc

Floating Point Usage

Usually, when a context switch occurs:

- the current state of a CPU’ s registers are saved to the stack of task being switch out
- the previously saved state of the CPU’ s registers are loaded from the stack of the task being switched in

However, ESP-IDF FreeRTOS implements Lazy Context Switching for the FPU (Floating Point Unit) registers of a CPU. In other words, when a context switch occurs on a particular core (e.g., CPU0), the state of the core’ s FPU registers are not immediately saved to the stack of the task getting switched out (e.g., Task A). The FPU’ s registers are left untouched until:

- A different task (e.g., Task B) runs on the same core and uses the FPU. This will trigger an exception that will save the FPU registers to Task A’ s stack.
- Task A gets scheduled to the same core and continues execution. Saving and restoring the FPU’ s registers is not necessary in this case.

However, given that tasks can be unpinned thus can be scheduled on different cores (e.g., Task A switches to CPU1), it is unfeasible to copy and restore the FPU’ s registers across cores. Therefore, when a task utilizes the FPU (by using a float type in its call flow), ESP-IDF FreeRTOS will automatically pin the task to the current core it is running on. This ensures that all tasks that use the FPU are always pinned to a particular core.

Furthermore, ESP-IDF FreeRTOS by default does not support the usage of the FPU within an interrupt context given that the FPU’ s register state is tied to a particular task.

备注: ESP targets that contain an FPU do not support hardware acceleration for double precision floating point arithmetic (double). Instead double is implemented via software hence the behavioral restrictions regarding the float type do not apply to double. Note that due to the lack of hardware acceleration, double operations may consume significantly more CPU time in comparison to float.

ESP-IDF FreeRTOS Single Core

Although ESP-IDF FreeRTOS is an SMP scheduler, some ESP targets are single core (such as the ESP32-S2 and ESP32-C3). When building ESP-IDF applications for these targets, ESP-IDF FreeRTOS is still used but the number of cores will be set to 1 (i.e., the CONFIG_FREERTOS_UNICORE will always be enabled for single core targets).

For multicore targets (such as the ESP32 and ESP32-S3), CONFIG_FREERTOS_UNICORE can also be set. This will result in ESP-IDF FreeRTOS only running on CPU0, and all other cores will be inactive.
Chapter 4. API

4.11 ESP-WIFI-MESH

Users should bear in mind that enabling `CONFIG_FREERTOS_UNICORE` is NOT equivalent to running Vanilla FreeRTOS. The additional API of ESP-IDF FreeRTOS can still be called, and the behavior changes of ESP-IDF FreeRTOS will incur a small amount of overhead even when compiled for only a single core.

4.11.1 概述

ESP-WIFI-MESH 是一套建立在 Wi-Fi 协议之上的网络协议。ESP-WIFI-MESH 允许分布在大范围区域内（室内和室外）的大量设备（下文简称节点）在同一个 WLAN（无线局域网）中相互连接。ESP-WIFI-MESH 具有自组网和自修复的特性，也就是说 mesh 网络可以自主地构建和维护。

本 ESP-WIFI-MESH 指南分为以下几个部分:

1. 简介
2. ESP-WIFI-MESH 概念
3. 建立网络
4. 管理网络
5. 数据传输
6. 信道切换
7. 性能
8. 更多注意事项

### 图 19: 传统 Wi-Fi 网络架构

传统基础设施 Wi-Fi 网络是一个 “单点对多点” 的网络。这种网络架构的中心节点为接入点 (AP)，其他节点 (station) 均与 AP 直接相连。其中，AP 负责各个 station 之间的仲裁和转发，一些 AP 还会通过路由
第4章 API

器与外部 IP 网络交换数据。在传统 Wi-Fi 网络架构中，1) 由于所有 station 均需与 AP 直接相连，不能距离 AP 太远，因此覆盖区域相对有限；2) 受到 AP 容量的限制，因此网络中允许的 station 数量相对有限，很容易超载。

图 20: ESP-WIFI-MESH 网络架构示意图

ESP-WIFI-MESH 与传统 Wi-Fi 网络的不同之处在于：网络中的节点不需要连接到中心节点，而是可以与相邻节点连接。各节点均负责相连节点的数据中继。由于无线受限于距离中心节点的位置，所有节点仍可互连。因此 ESP-WIFI-MESH 网络的覆盖区域更广。类似地，由于不再受限于中心节点的容量限制，ESP-WIFI-MESH 允许更多节点接入，也不易于超载。

4.11.3 ESP-WIFI-MESH 概念

术语

<table>
<thead>
<tr>
<th>术语</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>节点</td>
<td>任何属于或可以成为 ESP-WIFI-MESH 网络一部分的设备。</td>
</tr>
<tr>
<td>根节点</td>
<td>网络顶部的节点</td>
</tr>
<tr>
<td>子节点</td>
<td>如节点 X 连接至节点 Y，且 X 相较 Y 与根节点的距离更远（跨越的连接数量更多），则称 X 为 Y 的子节点。</td>
</tr>
<tr>
<td>父节点</td>
<td>与子节点对应的概念。</td>
</tr>
<tr>
<td>后裔节点</td>
<td>任何可以从根节点追踪到的节点。</td>
</tr>
<tr>
<td>兄弟节点</td>
<td>连接至同一个父节点的所有节点。</td>
</tr>
<tr>
<td>连接</td>
<td>AP 和 station 之间的传统 Wi-Fi 关联。ESP-WIFI-MESH 中的节点使用 station 接口与另一个节点的 SoftAP 接口产生关联，进而形成连接。连接包括 Wi-Fi 网络中的身份验证和关联过程。</td>
</tr>
<tr>
<td>上行连接</td>
<td>从节点到其父节点的连接。</td>
</tr>
<tr>
<td>下行连接</td>
<td>从父节点到其一个子节点的连接。</td>
</tr>
<tr>
<td>无线 hop</td>
<td>源节点和目标节点间无线连接路径中的一部分。多跳指遍历单个连接的数据包，多跳指遍历多个连接的数据包。</td>
</tr>
<tr>
<td>子网</td>
<td>子网指 ESP-WIFI-MESH 网络的一部分，包括一个节点及其所有后代节点。因此，根节点的子网包括 ESP-WIFI-MESH 网络中的所有节点。</td>
</tr>
<tr>
<td>MAC 地址</td>
<td>在 ESP-WIFI-MESH 网络中用于区别每个节点或路由器的唯一地址。</td>
</tr>
<tr>
<td>DS</td>
<td>分布式系统（外部 IP 网络）。</td>
</tr>
</tbody>
</table>
### 树型拓扑

ESP-WIFI-MESH 建立在传统 Wi-Fi 协议之上，可被视为一种将多个独立 Wi-Fi 网络组合为一个单一WLAN 网络的组网协议。在 Wi-Fi 网络中，station 在任何时候都仅限于与 AP 建立单个连接（上行连接），而 AP 则可以同时连接到多个 station（下行连接）。然而，ESP-WIFI-MESH 网络则允许节点同时充当 station 和 AP。因此，ESP-WIFI-MESH 中的节点可以使用其 SoftAP 接口建立多个下行连接，同时使用其 station 接口建立一个上行连接。这将自然产生一个由多层父子结构组成的树型网络拓扑结构。

![ESP-WIFI-MESH Tree Topology](image)

**图 21: ESP-WIFI-MESH 树型拓扑**

ESP-WIFI-MESH 是一个多跳网络，也就是说网络中的节点可以通过单跳或多跳向网络中的其他节点传送数据包。因此，ESP-WIFI-MESH 中的节点不仅传输自己的数据包，而且同时充当其他节点的中继。假设 ESP-WIFI-MESH 网络中的任何两个节点存在物理层上连接（通过单跳或多跳），则这两个节点可以进行通信。

**备注**: ESP-WIFI-MESH 网络中的大小（节点总数）取决于网络中允许的最大层级，以及每个节点可以具有的最大下行连接数。因此，这两个变量可用于配置 ESP-WIFI-MESH 网络的大小。

### 节点类型

- **根节点**: 指网络顶部的节点，是 ESP-WIFI-MESH 网络和外部 IP 网络之间的唯一接口。根节点直接连接至传统的 Wi-Fi 路由器，并在 ESP-WIFI-MESH 网络的节点和外部 IP 网络之间中继数据包。ESP-WIFI-MESH 网络中只能有一个根节点，且根节点的上行连接只能是路由器。如上图所示，节点 A 即为该 ESP-WIFI-MESH 网络的根节点。

- **叶子节点**: 指不允许拥有任何子节点（即无下行连接）的节点。因此，叶子节点只能传输或接收自己的数据包，但不能转发其他节点的数据包。如果节点处于 ESP-WIFI-MESH 网络的最大允许层级，则该节点将成为叶子节点。叶子节点不回再产生下行连接，这可以防止节点继续生成下行连接，从而确保网络层级不会超出限制。由于建立下行连接必须使用 SoftAP 接口，因此一些没有 SoftAP 接口的节点（仅有 station 接口）也将被分配为叶子节点。如上图所示，位于网络最外层的 L/M/N 节点即为叶子节点。

- **中间父节点**: 既不是根节点也不属于叶子节点的节点即为中间父节点。中间父节点必须有且仅有一个上行连接（即一个父节点），但可以具有 0 个或多个下行连接（即 0 个或多个子节点）。因此，中间父节点可以发送和接收自己的数据包，也可以转发其上行和下行连接的数据包。如上图所示，节点 B 到 J 即为中间父节点。注意，E/F/G/I/J 等没有下行连接的中间父节点并不等同于叶子节点，原因在于这些节点仍允许形成下行连接。
图 22: ESP-WIFI-MESH 节点类型

空闲节点：尚未加入网络的节点即为空闲节点。空闲节点将尝试与中间父节点形成上行连接，或者在有条件的情况下（参见自动根节点选择）成为一个根节点。如上图所示，K 和 O 节点即为空闲节点。

信标帧和 RSSI 阈值

ESP-WIFI-MESH 中能够形成下行连接的每个节点（即具有 SoftAP 接口）都会定期传输 Wi-Fi 信标帧。节点可以通过信标帧让其他节点检测自己的存在和状态。空闲节点将侦听信标帧以生成一个潜在父节点列表，并与其中一个潜在父节点形成上行连接。ESP-WIFI-MESH 使用“供应商信息元素”来存储元数据，例如：

- 节点类型（根节点、中间父节点、叶子节点、空闲节点）
- 节点当前所处的层级
- 网络中允许的最大层级
- 当前子节点数量
- 可接受的最大下行连接数量

潜在上行连接的信号强度可由潜在父节点信标帧的 RSSI 表示。为了防止节点形成弱上行连接，ESP-WIFI-MESH 采用了针对信标帧的 RSSI 阈值控制机制。如果节点检测到某节点的信标帧 RSSI 过低（即低于预设阈值），则会在尝试形成上行连接时忽略该节点。

上图（A侧）展示了 RSSI 阈值将如何影响空闲节点的候选父节点数量。

上图（B侧）展示了 RF 屏蔽物将如何降低潜在父节点的 RSSI。由于存在 RF 屏蔽物，节点 X 的 RSSI 高于阈值的区域显著减小。这会导致空闲节点忽略节点 X，即使从地理位置上看 X 就在空闲节点附近。相反，该空闲节点将从更远的地方找到一个 RSSI 更强的节点 Y 形成上行连接。

备注：事实上，ESP-WIFI-MESH 网络中的节点在 MAC 层仍可以接收所有的信标帧，但 RSSI 阈值控制功能可以过滤掉所有 RSSI 低于预设阈值的信标帧。
首选父节点

当一个空闲节点有多个候选父节点（潜在父节点）时，空闲节点将与其中的首选父节点形成上行连接。首选父节点基于以下条件确定：

- 候选父节点所处的层级
- 候选父节点当前具有的下行连接（子节点）数量

在网络中所处层级较浅的候选父节点（包括根节点）将优先成为首选父节点。这有助于在形成上行连接时控制 ESP-WIFI-MESH 网络中的总层级使之最小。例如，在位于第二层和第三层的候选父节点中选择时，位于第二层的候选父节点将始终优先成为首选父节点。

如果同一层上存在多个候选父节点，子节点最少的候选父节点将优先成为首选父节点。这有助于平衡同一层节点的下行连接数量。

图 23: RSSI 阈值的影响

图 24: 首选父节点选择
上图（A侧）展示了空闲节点G如何在B/C/D/E/F五个候选父节点中选择首选父节点：首先，B/C节点优于D/E/F节点，因为这两个节点所处的层级更深。其次，C节点优于B节点，因为C节点的下行连接数量（子节点数量）更少。

上图（B侧）展示了空闲节点G如何在根节点A和其他候选父节点中选择首选父节点：此时根节点A处于空闲节点G范围内（即空闲节点G接收到的根节点A信标帧RSSI强度高于阈值）。由于根节点A处于网络中层的最层，因此将高为首选父节点。

备注：用户还可以自行定义首选父节点的选择规则，也可以直接指定某个节点为首选父节点（见Mesh手动配置示例）。

路由表

ESP-WIFI-MESH网络中的每个节点将维护自己的路由表，并按路由表将数据包（请见ESP-WIFI-MESH数据包）沿正确的路线发送至正确的目标节点。某个特定节点的路由表将包含该节点的子网中所有节点的MAC地址，也包括该节点自己的MAC地址。每个路由表会划分为多个子路由表，与每个子节点的子网对应。

图25: ESP-WIFI-MESH路由表示例

以上图为例，节点B的路由表中将包含节点B到节点I的MAC地址（即相当于节点B的子网）。节点B的路由表可划分为节点C和G的子路由表，分别包含节点C到节点F的MAC地址，节点G到节点I的MAC地址。

ESP-WIFI-MESH利用路由表来使用以下规则进行转发：确定ESP-WIFI-MESH数据包应根据向上传转发还是向下转发。

1. 如果数据包的目标MAC地址处于当前节点的路由表中且不等于当前节点的MAC地址，则选择包含目标MAC地址的子路由表，并将数据包向下转发给子路由表对应的子节点。

2. 如果数据包的目标MAC地址不在当前节点的路由表内，则将数据包向上转发给当前节点的父节点，并重复执行该操作直至数据包达到目标地址。此步骤可重复至根节点（根节点包含整个网络的全部节点）。

备注：用户可以通过调用esp_mesh_get_routing_table()获取一个节点的路由表，调用esp_mesh_get_routing_table_size()获取一个路由表的大小，也可
### 4.11.4 建立网络

一般过程

#### 警告：
ESP-WIFI-MESH 正式开始构建网络前，必须确保网络中所有节点具有相同的配置（见 mesh_cfg_t）。每个节点必须配置相同 MESH 网络 ID、路由器配置和 SoftAP 配置。

ESP-WIFI-MESH 网络将首先选择根节点，然后逐层形成下行连接，直到所有节点均加入网络。网络的布局可能取决于诸如根节点选择、父节点选择和异步上电复位等因素。但简单来说，一个 ESP-WIFI-MESH 网络的构建过程可以概括为以下步骤：

1. **根节点选择**  根节点直接进行指定（见用户指定根节点）或通过选举由信号强度最强的节点担任（见自动根节点选择）。一旦选定，根节点将与路由器连接，并开始允许下行连接形成。如上图所示，节点 A 被选为根节点，因此节点 A 上行连接到路由器。

2. **第二层形成**  一旦根节点连接到路由器，根节点范围内的空闲节点将开始与根节点连接，从而形成第二层网络。一旦连接，第二层节点成为中间父节点（假设最大允许层级大于 2 层），并进而形成下一层。如上图所示，节点 B 到节点 D 都在根节点的连接范围内。因此，节点 B 到节点 D 将与根节点形成上行连接，并成为中间父节点。

3. **其余层形成**  剩余的空闲节点将与所处范围内的中间父节点连接，并形成新的层。一旦连接，根据网络的最大允许层级，空闲节点成为中间父节点或叶子节点。之后重复该步骤，直到网络中的所有空闲节点均加入网络或达到网络最大允许层级。如上图所示，节点 E/F/G 分别与节点 B/C/D 连接，并成为中间父节点。

图 26: ESP-WIFI-MESH 网络构建过程
## API

### 4. 限制树深度

为了防止网络超过最大允许层级，最大允许层级上的节点将在完成连接后成为叶子节点。这样一来，其他空闲节点将无法与这些最大允许层级上的叶子节点形成连接，因此不会超过最大允许层级。然而，如果空闲节点无法找到其他潜在父节点，则将无限期地保持空闲状态。如上图所示，网络的最大允许层级为四。因此，节点 H 在完成连接后将成为叶子节点，以防止任何下行连接的形成。

### 自动根节点选择

在自动模式下，根节点的选择取决于相对于路由器的信号强度。每个空闲节点将通过 Wi-Fi 信标帧发送自己的 MAC 地址和路由器 RSSI 值。MAC 地址可以表示网络中的唯一节点，而路由器 RSSI 值代表相对于路由器的信号强度。

此后，每个节点将同时扫描来自其他空闲节点的信标帧。如果节点检测到具有更强的路由器 RSSI 的信标帧，则节点将开始传输该信标帧的内容（相当于为这个节点投票）。经过最小迭代次数（可预先设置，默认为 10 次），将选举出路由器 RSSI 值最强的信标帧。

在达到预设迭代次数后，每个节点将单独计算其得票百分比（得票数/总票数）以确定它是否应该成为根节点。如果节点的得票百分比大于预设的阈值（默认为 90%），则该节点将成为根节点。

下图展示了在 ESP-WIFI-MESH 网络中，根节点的自动选择过程。

![图 27: 根节点选举示例](image)

1. 上电复位时，每个节点开始传输自己的信标帧（包括 MAC 地址和路由器 RSSI 值）。
2. 在多次传输和扫描迭代中，路由器 RSSI 最强的信标帧将在整个网络中传播。节点 C 具有最强的路由器 RSSI 值 (-10 dB)，因此它的信标帧将在整个网络中传播。所有参与选举的节点都给节点 C 投票，因此节点 C 的得票百分比为 100%。因此，节点 C 成为根节点，并与路由器连接。
3. 一旦节点 C 与路由器连接，节点 C 将成为节点 A/B/D/E 的首选父节点（即最浅的节点），并与其他节点连接。节点 A/B/D/E 将形成网络的第二层。
4. 节点 F 和节点 G 分别连接节点 D 和节点 E，并完成网络构建过程。

备注：用户可以通过 esp_mesh_set_attempts() 配置选举的最小迭代次数。用户应根据网络内的节点数量配置迭代次数（即 mesh 网络越大，所需的迭代次数越高）。
警告：得票百分比阈值也可以使用 `esp_mesh_set_vote_percentage()` 进行配置。得票百分比阈值过低可能导致同一 mesh 网络中两个或多个节点成为根节点，进而分化为多个 mesh 网络。如果发生这种情况，ESP-WIFI-MESH 具有内部机制，可自主解决 根节点冲突。这些具有多个根节点的网络将围绕一个根节点形成一个网络。然而，两个或多个路由器 SSID 相同但路由器 BSSID 不同的根节点冲突无法解决。

用户指定根节点

根节点也可以由用户指定，即直接让指定的根节点与路由器连接，并放弃选举过程。当根节点指定后，网络内的所有其他节点也必须放弃选举过程，以防止根节点冲突的发生。下图展示了在 ESP-WIFI-MESH 网络中，根节点的手动选择过程。

![图 28: 根节点指定示例（根节点 = A，最大层级 = 4）](image)

1. 节点 A 是由用户指定的根节点，因此直接与路由器连接。此时，所有其他节点放弃选举过程。
2. 节点 C 和节点 D 将节点 A 选为自己的首选父节点，并与其形成连接。这两个节点将形成网络的第二层。
3. 类似地，节点 B 和节点 E 将与节点 C 连接，节点 F 将与节点 D 连接。这三个节点将形成网络的第三层。
4. 节点 G 将与节点 E 连接，形成网络的第四层。然而，由于该网络的最大允许层级已配置为 4，因此节点 G 将成为叶子节点，以防止形成任何新层。

备注：一旦指定根节点，该根节点应调用 `esp_mesh_set_parent()` 使其直接与路由器连接。类似地，所有其他节点都应该调用 `esp_mesh_fix_root()` 放弃选举过程。

选择父节点

默认情况下，ESP-WIFI-MESH 具有可以自组网的特点，也就是每个节点都可以自主选择与其形成上行连接的潜在父节点。自主选择出的父节点被称为首选父节点。用于选择首选父节点的标准旨在减少 ESP-WIFI-MESH 网络的层级，并平衡各个潜在父节点的下行连接数（参见 首选父节点）。
不过，ESP-WIFI-MESH 也允许用户禁用自组网功能，即允许用户自定义父节点选择标准，或直接指定某个节点为父节点（见：Mesh 手动组网示例）。

异步上电复位

ESP-WIFI-MESH 网络构建可能会受到节点上电顺序的影响。如果网络中的某些节点为异步上电（即相隔几分钟上电），网络的最终结构可能与所有节点同步上电时的理想情况不同。延迟上电的节点将遵循以下规则：

规则 1：如果网络中已存在根节点，则延迟节点不会尝试选举成为新的根节点。即使自身的路由器 RSSI 更强。反而，延迟节点与任何其他空闲节点无异，将通过与首选父节点连接来加入网络。如果该延迟节点为用户指定的根节点，则网络中的所有其他节点将保持空闲状态，直到延迟节点完成上电。

规则 2：如果延迟节点形成上行连接，并成为中间父节点，则后续也可能成为其他节点（即其他更浅的节点）的新首选父节点。此时，其他节点切换上行连接至该延迟节点（见

规则 3：如果空闲节点的指定父节点上电延迟了，则该空闲节点在没有找到指定父节点前不会尝试形成任何上行连接。空闲节点将无限制地保持空闲，直到其指定的父节点上电完成。

下方示例展示了异步上电对网络构建的影响。

图 29: 网络构建（异步电源）示例

1. 节点 A/C/D/F/G/H 同步上电，并通过广播其 MAC 地址和路由器 RSSI 开始选举根节点。节点 A 的 RSSI 最强，因此当选为根节点。
2. 一旦节点 A 成为根节点，其余的节点就开始与其首选父节点逐层形成上行连接，并最终形成一个具有五层的网络。
3. 节点 B/E 由于存在上电延迟，因此即使路由器 RSSI 比节点 A 更强（-20 dB 和 -10 dB）也不会尝试成为根节点。相反，这两个上电延迟节点均将与对应的首选父节点 A 和 C 形成上行连接。加入网络后，节
点 B/E 均将成为中间父节点。

4. 节点 B 由于所处层级变化（现为第二层）而成为新的首选父节点，因此节点 D/G 将切换其上行连接从而选择新的首选父节点。由于切换的发生，最终的网络层级从原来的五层减少至三层。

同步上电：如果所有节点均同步上电，节点 E (-10 dB) 由于路由器 RSSI 最强而成为根节点。此时形成的网络结构将与正常上电的情况截然不同。但是，如果用户手动切换根节点，则仍可以达到同步上电的网络结构（请见 esp_mesh_waive_root()）。

备注：从某种程度上，ESP-WIFI-MESH 可以自动修复部分因异常上电引起的父节点选择的偏差（请见父节点切换）

环路避免、检测与处理

环路是指特定节点与其后续节点（特定节点子网中的节点）形成上行连接的情况。因此产生的循环连接路径将打破 mesh 网络的拓扑结构。ESP-WIFI-MESH 的节点在选择父节点时将主动排除路由表（见路由表）中的节点，从而避免与其子网中的节点建立上行连接并形成环路。

在存在环路的情况下，ESP-WIFI-MESH 可利用链路验证机制和容量控制机制来检测环路的产生。因与子节点建立上行连接而导致环路形成的父节点将通知其子节点环路的存在，并主动断开连接。

4.11.5 管理网络

作为一个自修复网络，ESP-WIFI-MESH 可以检测并修正网络路由中的故障。当具有一个或多个子节点的父节点断开或父节点与其子节点之间的连接不时稳定时，会发生故障。ESP-WIFI-MESH 中的子节点将自主选择一个新的父节点，并与其形成上行连接，以维持网络连接。ESP-WIFI-MESH 可以处理根节点故障和中间父节点故障。

根节点故障

如果根节点断开，则与其连接的节点（第二层节点）将及时检测到该根节点故障。第二层节点将主动尝试与根节点重连，但在多次尝试失败后，第二层节点将启动新一轮的根节点选举。第二层中 RSSI 最强的节点将当选为新的根节点。而剩余的第二层节点将与新的根节点（如果不在范围内的话，也可与相邻父节点连接）形成上行连接。

如果根节点和下多层的节点（例如根节点、第二层节点和第三层节点）同时断开，则位于最底层的仍在正常工作的节点将发起根节点选举。下方示例展示了网络从根节点断开故障中进行自修复。

![图 30: 根节点故障的自修复示意](image)

1. 节点 C 是网络的根节点。节点 A/B/D/E 是连接到节点 C 的第二层节点。
2. 节点 C 断开。在多次重连尝试失败后，第二层节点开始通过广播其路由器 RSSI 开始新一轮的选举。此时，节点 B 的路由器 RSSI 最强。

3. 节点 B 被选为根节点，并开始接受下行连接。剩余的第二层节点 A/D/E 形成与节点 B 的上行连接，因此网络已经恢复，并且可以继续正常运行。

备注：如果是手动指定的根节点断开，则无法进行自动修复。任何节点不会在存在指定根节点的情况下开始选举过程。

中间父节点故障

如果中间父节点断开，则与之断开的子节点将主动尝试与该父节点重连。在多次重连尝试失败后，每个子节点开始扫描潜在父节点（请见标题和 RSSI 阈值）。

如果存在其他可用的潜在父节点，每个子节点将分别给予自己一个新的首选父节点（请见自动父节点），并与它形成上行连接。如果特定子节点没有其他潜在的父节点，则将无限期地保持空闲状态。

下方示例展示了网络中中间父节点断开故障中进行自修复。

![图31: 中间父节点故障的自修复](image)

1. 网络中存在节点 A 至 G。

2. 节点 C 断开。节点 F/G 检测到节点 C 的断开故障，并尝试与节点 C 重新连接。在多次重连尝试失败后，节点 F/G 将开始选择新的首选父节点。

3. 节点 G 因其区域内不存在父节点而暂时保持空闲。节点 F 的范围内有 B 和 E 两个节点，但节点 B 因为所处层级更浅而当选新的父节点。节点 F 将与节点 B 连接，并成为一个中间父节点，节点 G 将于节点 F 相连。这样一来，网络已经恢复了，但结构发生了变化（网络层级增加了 1 层）。

备注：如果子节点的父节点已被指定，则子节点不会尝试与其他潜在父节点连接。此时，该子节点将无限期地保持空闲状态。

根节点切换

除非根节点断开，否则 ESP-WIFI-MESH 不会自动切换根节点。即使根节点的路由器 RSSI 降低至必须断开的情况，根节点也将保持不变。根节点切换是指明确宣布新选举过程的行为，即具有更强路由器 RSSI 的节点选为新的根节点。这可以用于应对根节点性能降低的情况。

要触发根节点切换，当前根节点必须明确调用 esp_mesh_waive_root() 以触发新的选举。当下根节点将指示网络中的所有节点开始发送并扫描信标帧（见自动根节点选择），但与此同时一直保持联网（即...
不会变成空闲节点。如果另一个节点收到的票数超过当前根节点，则将启动根节点切换过程，否则根节点将保持不变。

新选出的根节点向当前的根节点发送切换请求，而原先的根节点将返回一个应答通知，表示已经准备好切换。一旦接收到来自新根节点的应答，新选出的根节点将与其父节点断开连接，并迅速与路由器形成上行连接，进而成为网络的新根节点。原先的根节点将断开与路由器的连接，并与此同时保持其所有下行连接并进入空闲状态。之前的根节点将开始扫描潜在的父节点并选择首选父节点。

下图说明了根节点切换的示例。

切换根节点示例

1. 节点 C 是当前的根节点，但路由器 RSSI 值 (-85 dB) 降低至较低水平。此时，新的选举过程被触发了。所有节点开始传输和扫描信号强度（此时仍保持连接）。

2. 经过多轮传输和扫描后，节点 B 被选为新的根节点。节点 B 向节点 C 发送了一个切换请求，节点 C 回复一个应答。

3. 节点 B 与其父节点断开连接，并与路由器连接，成为网络中的新根节点。节点 C 与路由器断开连接，进入空闲状态，并开始扫描并选择新的首选父节点。节点 C 在整个过程中仍保持其所有的下行连接。

4. 节点 C 选择节点 B 作为其的首选父节点，与之形成上行连接，并成为一个第二层节点。由于节点 C 仍保持相同的子网，因此根节点切换后的网络结构没有变化。然后，由于切换的发生，节点 C 子网中每个节点的所处层级均增加了一层。如果根节点切换过程中产生了新的根节点，则父节点切换可以随后调整网络结构。

备注：根节点切换必须要求选举，因此只有在使用自组网 ESP-WIFI-MESH 网络时才支持。换句话说，如果使用指定的根节点，则不能进行根节点切换。

父节点切换

父节点切换是指一个子节点将其上行连接切换到更浅一层的另一个父节点。父节点切换是自动的，这意味着如果较浅层出现了可用的潜在父节点（因“异步上电复位”产生），子节点将自动更改其上行连接。

所有潜在的父节点将定期发送信标帧（参见信标帧和 RSSI 阈值），从而允许子节点扫描较浅层的父节点的可用性。由于父节点切换，自组网 ESP-WIFI-MESH 网络可以动态调整其网络结构，以确保每个连接均具有良好的 RSSI 值，并且网络中的层级最小。
4.11.6 数据传输

ESP-WIFI-MESH 数据包

ESP-WIFI-MESH 网络使用 ESP-WIFI-MESH 数据包传输数据。ESP-WIFI-MESH 数据包完全包含在 Wi-Fi 数据帧中。ESP-WIFI-MESH 网络中的数据传输将涉及通过不同 Wi-Fi 数据帧在每个无线跳上传输的单个 ESP-WIFI-MESH 数据包。

下图显示了 ESP-WIFI-MESH 数据包的结构及其与 Wi-Fi 数据帧的关系。

![ESP-WIFI-MESH 数据包结构图](image)

图 32: ESP-WIFI-MESH 数据包

ESP-WIFI-MESH 数据包的 报头 包含源节点和目标节点的 MAC 地址。 选项 (option) 字段包含有关特殊类型的 ESP-WIFI-MESH 数据包的信息，例如组传输或来自外部 IP 网络的数据包（请参阅 MESH_OPT_SEND_GROUP 和 MESH_OPT_RECV_DS_ADDR）。

ESP-WIFI-MESH 数据包的 有效载荷 包含实际的应用数据。该数据可以为原始二进制数据，也可以是使用 HTTP、MQTT 和 JSON 等应用层协议的编码数据（请见：mesh_proto_t）。

备注：当向外部 IP 网络发送 ESP-WIFI-MESH 数据包时，报头的目标地址字段将包含目标服务器的 IP 地址和端口号，而不是节点的 MAC 地址（请见：mesh_addr_t）。此外，根节点将处理外发 TCP/IP 数据包的形成。

组控制和组播

组播功能允许将单个 ESP-WIFI-MESH 数据包同时发送给网络中的多个节点。ESP-WIFI-MESH 中的组播可以通过“指定一个目标节点列表”或“预配置一个节点组”来实现。这两种组播方式均需调用 esp_mesh_send() 实现。

如果通过“指定目标节点列表”实现组播，用户必须首先将 ESP-WIFI-MESH 数据包的目标地址设置为 组播组地址（例如 01:00:5E:xx:xx:xx）。这表明 ESP-WIFI-MESH 数据包是一个拥有一组地址的组播数据包，且该地址应该从报头选项中获得。然后，用户必须将目标节点的 MAC 地址列为选项（请见：mesh_opt_t 和 MESH_OPT_SEND_GROUP）。这种组播方法不需要进行提前设置，但由于每个目标节点的 MAC 地址均需列为报头的选项字段，因此会产生大量开销数据。

分组组播允许 ESP-WIFI-MESH 数据包被发送到一个预先配置的节点组。每个分组都有一个唯一的 ID 标识。用户可通过 esp_mesh_set_group_id() 将节点加入一个组。分组组播需要将 ESP-WIFI-MESH 数据包的目标地址设置为目标组的 ID，还必须设置 MESH_DATA_GROUP 标志位。分组组播产生的开销较小，但必须提前将节点加入分组中。

备注：在组播期间，网络中的所有节点在 MAC 层都会收到 ESP-WIFI-MESH 数据包。然而，不包括在 MAC 地址列表或目标组中的节点将简单地过滤掉这些数据包。
广播

广播功能允许将单个 ESP-WIFI-MESH 数据包同时发送给网络中的所有节点。每个节点可以将一个广播包转发至其所有上行和下行连接，使得数据包尽可能快地在整个网络中传播。但是，ESP-WIFI-MESH 利用以下方法来避免在广播期间浪费带宽。

1. 当中间父节点收到来自其父节点的广播包时，它会将该数据包转发给自己的各个子节点，同时为自己保存一份数据包的副本。
2. 当中间父节点是广播的源节点时，它会将该数据包向上发送至其父节点，并向下发送给自己的各个子节点。
3. 当中间父节点接收到一个来自其子节点的广播包时，它会将该数据包转发给其父节点和其余子节点，同时为自己保存一份数据包的副本。
4. 当叶子节点是广播的源节点时，它会直接将该数据包发送至其父节点。
5. 当根节点是广播的源节点时，它会将该数据包发送至自己的所有子节点。
6. 当根节点收到来自其子节点的广播包时，它会将该数据包转发给其余子节点，同时为自己保存一份数据包的副本。
7. 当节点接收到一个源地址与自身 MAC 地址匹配的广播包时，它会将该广播包丢弃。
8. 当中间父节点收到一个来自其父节点的广播包时（该数据包最初来自该父节点的一个子节点），它会将该广播包丢弃。

上行流量控制

ESP-WIFI-MESH 依赖父节点来控制其直接子节点的上行数据流。为了防止父节点的消息缓冲区上限情况下溢出，父节点将为每个子节点分配一个称为 接收窗口 的上行传输配额。每个子节点均必须申请接收窗口才允许进行上行传输。接收窗口的大小可以动态调整。完成从子节点到父节点的上行传输包括以下步骤：

1. 在每次传输之前，子节点向其父节点发送窗口请求。窗口请求中包括一个序号，与子节点的待传输数据包相对应。
2. 父节点接收窗口请求，并将序号与子节点发送的前一个数据包的序号进行比较，用于计算返回给子节点的接收窗口大小。
3. 子节点根据父节点指定的窗口大小发送数据包。如果子节点的接收窗口耗尽，它必须通过发送请求获得另一个接收窗口，然后才允许继续发送。

备注：ESP-WIFI-MESH 不支持任何下行流量控制。

警告：由于父节点切换，数据包可能会在上行传输期间丢失。

由于根节点是通信外部 IP 网络的唯一接口，因此下行节点必须了解根节点与外部 IP 网络的连接状态。否则，下节点可能会尝试向一个已经与 IP 网络断开连接的根节点发送数据，从而造成不必要的传输和数据包丢失。ESP-WIFI-MESH 可以基于监测根节点和外部 IP 网络的连接状态，提供一种稳定外发数据吞吐量的机制。根节点可以通过调用 esp_mesh_post_toDS_state() 将自身与外部 IP 网络的连接状态广播给所有其他节点。

双向数据流

下图展示了 ESP-WIFI-MESH 双向数据流涉及的各种网络层。

由于使用路由表，ESP-WIFI-MESH 能够在 mesh 层中完全处理数据包的转发。TCP/IP 层仅与 mesh 网络的根节点有关，可帮助根节点与外部 IP 网络的数据包传送。
4.11.7 信道切换

背景

在传统的 Wi-Fi 网络中，信道代表预设的频率范围。在基础设施基本服务集 (BSS) 中，工作 AP 及与之相连的 station 必须处于传输信标的网络信道（1到14）中。物理上相邻的 BSS 使用相同的工作信道会导致干扰产生和性能下降。

为了允许 BSS 适应不断变化的物理层条件并保持性能，Wi-Fi 网络中增加了网络信道切换的机制。网络信道切换是将 BSS 移至新的工作信道，并同时最大限度地减少期间对 BSS 的影响。然而，我们应该认识到，网络信道切换可能不会成功，无法将原信道中的所有 station 均移动至新的信道。

在基础设施 Wi-Fi 网络中，网络信道切换由 AP 触发，目的是将该 AP 及与之相连的所有 station 同步切换到新的信道。网络信道切换是通过在 AP 的周期性发送信标帧内嵌入一个信道切换宣告 (CSA) 元素来实现的。在网络信道切换前，该 CSA 元素用于向所有连接的 station 广播有关即将发生的网络信道切换，并且将包含在多个信标帧中。

一个 CSA 元素包含有关 新信道号 和 信道切换计数 的信息。其中，信道切换计数指示在网络信道切换之前剩余的信标帧间隔 (TBTT) 数量。因此，信道切换计数依每个信标帧递减，并且允许与之连接的 station 与 AP 同步进行信道切换。

ESP-WIFI-MESH 网络信道切换

ESP-WIFI-MESH 网络信道切换还利用包含 CSA 元素的信标帧。然而，ESP-WIFI-MESH 作为一个多跳网络，其信标帧可能无法到达网络中的所有节点（这与单跳网络不同）。因此信道切换过程更加复杂。因此，ESP-WIFI-MESH 网络依赖于通过节点转发 CSA 元素，从而实现在整个网络中的传播。

当具有一个或多个子节点的中间父节点接收到包含 CSA 元素的信标帧时，该节点会将该元素除在其下一个发送的信标帧（即具有相同的 新信道号 和 信道切换计数）中，从而实现该 CSA 元素的转发。鉴于 ESP-WIFI-MESH 网络中的所有节点都接收到相同的 CSA 元素，这些节点可以使用 信道切换计数 来同步其信道切换，但也会经历因 CSA 元素转发造成的延迟。

ESP-WIFI-MESH 网络信道切换可以由路由器或根节点触发。

根节点触发 由根节点触发的信道切换只能在 ESP-WIFI-MESH 网络未连接到路由器时才会发生。通过调用 esp_mesh_switch_channel() 函数，根节点将设置一个初始 信道切换计数值，并开始在其信标帧中包含 CSA 元素。接着，每个 CSA 元素将抵达第二层节点，并通过第二层节点自己的信标帧继续进行向下转发。
网络信道切换的影响

- 由于 ESP-WIFI-MESH 网络信道切换与路由器的信道切换不同步，ESP-WIFI-MESH 网络和路由器之间会出现临时信道切换。

  - ESP-WIFI-MESH 网络的信道切换时间取决于 ESP-WIFI-MESH 网络的信标间隔和根节点的自定义信道切换计数值。
  - 在 ESP-WIFI-MESH 网络切换期间，信道差异将阻止根节点和路由器之间的任何数据交换。
  - 在 ESP-WIFI-MESH 网络中，根节点和中间节点将请求与其连接的子节点停止传输，直至信道切换发生（通过将 CSA 元素的信道切换模式字段设置为 1）。
  - 频繁的路由器信道切换可能会降低 ESP-WIFI-MESH 网络的性能。请注意，这可能是由 ESP-WIFI-MESH 网络本身造成的（例如由于 ESP-WIFI-MESH 网络的无线介质争用等原因）。此时，用户应禁用路由器信道的自主信道切换，并直接指定一个信道。

- 当存在临时信道差异时，根节点从技术上来说仍保持连接至路由器。

  - 如果根节点经过一定数量信标间隔仍无法接收到信标帧或探测来自路由器的响应，则会断开连接。
  - 断开连接后，根节点将自动重新扫描所有信道以确定是否存在路由器。

- 如果根节点无法接收任何路由器的 CSA 信标帧（例如短暂的路由器切换时间），则路由器将在没有 ESP-WIFI-MESH 网络的情况下执行信道切换。

  - 在路由器切换信道后，根节点将不再能够接收路由器的信标帧和探测响应，并导致在一定数量的信标间隔后断开连接。
  - 在断开连接后，根节点将重新所有信道，寻找路由器。
  - 根节点将在整个过程中维护与之相连的下行连接。

备注：虽然 ESP-WIFI-MESH 网络信道切换的目的是将网络内的所有节点移动到新的工作信道，但也应该认识到，信道切换可能无法成功移动所有节点（比如由于节点故障等原因）。
存在根节点：根节点的存在也会影响是否允许信道或路由器切换。
下表说明了在不同参数/条件组合下是否允许信道切换和路由器切换。请注意，X 代表参数“不关心”。

<table>
<thead>
<tr>
<th>预设信道</th>
<th>允许信道切换</th>
<th>预置路由器</th>
<th>允许路由器切换</th>
<th>存在根节点</th>
<th>允许切换？</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>X</td>
<td>N</td>
<td>X</td>
<td>X</td>
<td>信道与路由器</td>
</tr>
<tr>
<td>N</td>
<td>X</td>
<td>Y</td>
<td>N</td>
<td>X</td>
<td>仅信道</td>
</tr>
<tr>
<td>N</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td>X</td>
<td>信道与路由器</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>X</td>
<td>N</td>
<td>仅路由器</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>X</td>
<td>Y</td>
<td>信道与路由器</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>X</td>
<td>仅信道</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>仅路由器</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>信道与路由器</td>
</tr>
</tbody>
</table>

4.11.8 性能

ESP-WIFI-MESH 网络的性能可以基于以下多个指标进行评估：

组网时长：从头开始构建 ESP-WIFI-MESH 网络所需的总时长。

修复时间：从网络检测到节点断开到执行适当操作（例如生成新的根节点或形成新的连接等）以修复网络所需的时间。

每跳延迟：数据每经过一次无线 hop 而经历的延迟，即从父节点向子节点（或从子节点向父节点）发送一个数据包所需的时间。

网络节点容量：ESP-WIFI-MESH 网络可以同时支持的节点总数。该指标取决于节点可以接受到的最大下行连接数和网络中允许的最大层级。

ESP-WIFI-MESH 网络的常见性能指标如下表所示：

- 组网时长：< 60 秒
- 修复时间
  - 根节点断开：< 10 秒
  - 子节点断开：< 5 秒
- 每条延迟：10 到 30 毫秒

备注：上述性能指标的测试条件见下。

- 测试设备数量：100
- 最大允许下行连接数量：6
- 最大允许层级：6

备注：吞吐量取决于数据包错误率和 hop 数量。

备注：根节点访问外部 IP 网络的吞吐量直接受到 ESP-WIFI-MESH 网络中节点数量和路由器带宽的影响。

备注：用户应注意，ESP-WIFI-MESH 网络的性能与网络配置和工作环境密切相关。
4.11.9 更多注意事项

- 数据传输使用 Wi-Fi WPA2-PSK 加密
- Mesh 网络 IE 使用 AES 加密

本文图片中使用的路由器与互联网图标来自 www.flaticon.com 的 Smashicons。

4.12 Event Handling

Several ESP-IDF components use events to inform application about state changes, such as connection or disconnection. This document gives an overview of these event mechanisms.

4.12.1 Wi-Fi, Ethernet, and IP Events

Before the introduction of esp_event library, events from Wi-Fi driver, Ethernet driver, and TCP/IP stack were dispatched using the so-called legacy event loop. The following sections explain each of the methods.

esp_event Library Event Loop

esp_event library is designed to supersede the legacy event loop for the purposes of event handling in ESP-IDF. In the legacy event loop, all possible event types and event data structures had to be defined in system_event_id_t enumeration and system_event_info_t union, which made it impossible to send custom events to the event loop, and use the event loop for other kinds of events (e.g. Mesh). Legacy event loop also supported only one event handler function, therefore application components could not handle some of Wi-Fi or IP events themselves, and required application to forward these events from its event handler function.

See esp_event library API reference for general information on using this library. Wi-Fi, Ethernet, and IP events are sent to the default event loop provided by this library.

Legacy Event Loop

This event loop implementation is started using esp_event_loop_init() function. Application typically supplies an event handler, a function with the following signature:

```c
esp_err_t event_handler(void *ctx, system_event_t *event)
{

}
```

Both the pointer to event handler function, and an arbitrary context pointer are passed to esp_event_loop_init().

When Wi-Fi, Ethernet, or IP stack generate an event, this event is sent to a high-priority event task via a queue. Application-provided event handler function is called in the context of this task. Event task stack size and event queue size can be adjusted using CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE and CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE options, respectively.

Event handler receives a pointer to the event structure (system_event_t) which describes current event. This structure follows a tagged union pattern: event_id member indicates the type of event, and event_info member is a union of description structures. Application event handler will typically use switch(event->event_id) to handle different kinds of events.

If application event handler needs to relay the event to some other task, it is important to note that event pointer passed to the event handler is a pointer to temporary structure. To pass the event to another task, application has to make a copy of the entire structure.
### Event IDs and Corresponding Data Structures

<table>
<thead>
<tr>
<th>Event ID (legacy event ID)</th>
<th>Event data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wi-Fi</strong></td>
<td></td>
</tr>
<tr>
<td>WIFI_EVENT_WIFI_READY (SYSTEM_EVENT_WIFI_READY)</td>
<td>n/a</td>
</tr>
<tr>
<td>WIFI_EVENT_SCAN_DONE (SYSTEM_EVENT_SCAN_DONE)</td>
<td>wifi_event_sta_scan_done_t</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_START (SYSTEM_EVENT_STA_START)</td>
<td>n/a</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_STOP (SYSTEM_EVENT_STA_STOP)</td>
<td>n/a</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_CONNECTED (SYSTEM_EVENT_STA_CONNECTED)</td>
<td>wifi_event_sta_connected_t</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_DISCONNECTED (SYSTEM_EVENT_STA_DISCONNECTED)</td>
<td>wifi_event_sta_disconnected_t</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_AUTHMODE_CHANGE (SYSTEM_EVENT_STA_AUTHMODE_CHANGE)</td>
<td>wifi_event_sta_authmode_change_t</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_WPS_ER_SUCCESS (SYSTEM_EVENT_STA_WPS_ER_SUCCESS)</td>
<td>n/a</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_WPS_ER_FAILED (SYSTEM_EVENT_STA_WPS_ER_FAILED)</td>
<td>wifi_event_sta_wps_fail_reason_t</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_WPS_ER_TIMEOUT (SYSTEM_EVENT_STA_WPS_ER_TIMEOUT)</td>
<td>n/a</td>
</tr>
<tr>
<td>WIFI_EVENT_STA_WPS_ER_PIN (SYSTEM_EVENT_STA_WPS_ER_PIN)</td>
<td>wifi_event_sta_wps_er_pin_t</td>
</tr>
<tr>
<td>WIFI_EVENT_AP_START (SYSTEM_EVENT_AP_START)</td>
<td>n/a</td>
</tr>
<tr>
<td>WIFI_EVENT_AP_STOP (SYSTEM_EVENT_AP_STOP)</td>
<td>n/a</td>
</tr>
<tr>
<td>WIFI_EVENT_AP_STACONNECTED (SYSTEM_EVENT_AP_STACONNECTED)</td>
<td>wifi_event_ap_staconnected_t</td>
</tr>
<tr>
<td>WIFI_EVENT_AP_STADISCONNECTED (SYSTEM_EVENT_AP_STADISCONNECTED)</td>
<td>wifi_event_ap_stadisconnected_t</td>
</tr>
<tr>
<td>WIFI_EVENT_AP_PROBEREQRECVED (SYSTEM_EVENT_AP_PROBEREQRECVED)</td>
<td>wifi_event_ap_probe_req_rx_t</td>
</tr>
<tr>
<td><strong>Ethernet</strong></td>
<td></td>
</tr>
<tr>
<td>ETHERNET_EVENT_START (SYSTEM_EVENT_ETH_START)</td>
<td>n/a</td>
</tr>
<tr>
<td>ETHERNET_EVENT_STOP (SYSTEM_EVENT_ETH_STOP)</td>
<td>n/a</td>
</tr>
<tr>
<td>ETHERNET_EVENT_CONNECTED (SYSTEM_EVENT_ETH_CONNECTED)</td>
<td>n/a</td>
</tr>
<tr>
<td>ETHERNET_EVENT_DISCONNECTED (SYSTEM_EVENT_ETH_DISCONNECTED)</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>IP</strong></td>
<td></td>
</tr>
<tr>
<td>IP_EVENT_STA_GOT_IP (SYSTEM_EVENT_STA_GOT_IP)</td>
<td>ip_event_got_ip_t</td>
</tr>
<tr>
<td>IP_EVENT_STA_LOST_IP (SYSTEM_EVENT_STA_LOST_IP)</td>
<td>n/a</td>
</tr>
<tr>
<td>IP_EVENT_AP_STAIPASSIGNED (SYSTEM_EVENT_AP_STAIPASSIGNED)</td>
<td>n/a</td>
</tr>
<tr>
<td>IP_EVENT_STA_GOT_IP6 (SYSTEM_EVENT_STA_GOT_IP6)</td>
<td>ip_event_got_ip6_t</td>
</tr>
<tr>
<td>IP_EVENT_ETH_GOT_IP (SYSTEM_EVENT_ETH_GOT_IP)</td>
<td>ip_event_got_ip_t</td>
</tr>
<tr>
<td>IP_EVENT_ETH_LOST_IP (SYSTEM_EVENT_ETH_LOST_IP)</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**4.12.2 Mesh Events**

ESP-WIFI-MESH uses a system similar to the Legacy Event Loop to deliver events to the application. See 系统事件 for details.
4.12.3 Bluetooth Events

Various modules of the Bluetooth stack deliver events to applications via dedicated callback functions. Callback functions receive the event type (enumerated value) and event data (union of structures for each event type). The following list gives the registration API name, event enumeration type, and event parameter type.

- **BLE GAP**: esp_ble_gap_register_callback(), esp_gap_ble_cb_event_t, esp_ble_gap_cb_param_t.
- **BT GAP**: esp_bt_gap_register_callback(), esp_bt_gap_cb_event_t, esp_bt_gap_cb_param_t.
- **GATT**: esp_ble_gattc_register_callback(), esp_ble_gattc_cb_event_t, esp_ble_gattc_cb_param_t.
- **GATTS**: esp_ble_gatts_register_callback(), esp_ble_gatts_cb_event_t, esp_ble_gatts_cb_param_t.
- **SPP**: esp_spp_register_callback(), esp_spp_cb_event_t, esp_spp_cb_param_t.
- **Blufi**: esp_blufi_register_callbacks(), esp_blufi_cb_event_t, esp_blufi_cb_param_t.
- **A2DP**: esp_a2d_register_callback(), esp_a2d_cb_event_t, esp_a2d_cb_param_t.
- **AVRC**: esp_avrc_ct_register_callback(), esp_avrc_ct_cb_event_t, esp_avrc廖cb_param_t.
- **HFP Client**: esp_hf_client_register_callback(), esp_hf_client_cb_event_t, esp_hf_client_cb_param_t.
- **HFP AG**: esp_bt_hf_register_callback(), esp_hf_cb_event_t, esp_hf_cb_param_t.

4.13 严重错误

4.13.1 概述

在某些情况下，程序并不会按照我们的预期运行，在 ESP-IDF 中，这些情况包括：

- CPU 异常：非法指令，加载/存储时的内存对齐错误，加载/存储时的访问权限错误，双重异常。
- 系统级检查错误：
  - 中断看门狗 超时
  - 任务看门狗 超时（只有开启CONFIG_ESP_TASK_WDT_PANIC 后才会触发严重错误）
  - 高速缓存访问错误
  - 掉电检测事件
  - 堆栈溢出
  - 堆栈缓冲保护检查
  - 堆完整性检查
  - 未定义行为清理器 (UBSAN) 检查
- 使用 assert, configASSERT 等类似的宏断言失败。

本指南会介绍 ESP-IDF 中这类错误的处理流程，并给出对应的解决建议。

4.13.2 紧急处理程序

概述 中列举的所有错误都会由 紧急处理程序 (Panic Handler) 负责处理。

紧急处理程序首先会将出错原因打印到控制台，例如 CPU 异常的错误信息通常会类似于

Guru Meditation Error: Core 0 panic'ed (IllegalInstruction). Exception was __unhandled.

对于一些系统级检查错误（如中断看门狗超时，高速缓存访问错误等），错误信息会类似于
Guru Meditation Error: Core 0 panic'ed (Cache disabled but cached memory region accessed). Exception was unhandled.

不管哪种情况，错误原因都会被打印在括号中。请参阅Guru Meditation错误以查看所有可能的出错原因。紧急处理程序接下来的行为将取决于CONFIG_ESP_SYSTEM_PANIC的设置，支持的选项包括：

- 打印CPU寄存器，然后重启(CONFIG_ESP_SYSTEM_PANIC_PRINT_REBOOT) - 默认选项
- 打印系统发生异常时CPU寄存器的值。打印回溯，最后重启芯片。
- 打印CPU寄存器，然后暂停(CONFIG_ESP_SYSTEM_PANIC_PRINT_HALT)与上一个选项类似，但不会重启，而是选择暂停程序的运行。重启程序需要外部执行复位操作。
- 静默重启(CONFIG_ESP_SYSTEM_PANIC_SILENT_REBOOT)不打印CPU寄存器的值，也不打印回溯，立即重启芯片。
- 调用GDB Stub (CONFIG_ESP_SYSTEM_PANIC_GDBSTUB)启动GDB服务器，通过控制台UART接口与GDB进行通信。该选项只提供只读调试或者事后调试，详细信息请参阅GDBStub。
- 调用动态GDB Stub (CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME)启动GDB服务器，通过控制台UART接口与GDB进行通信。该选项允许用户在程序运行时对其进行调试、设置断点和改变其执行方式等，详细信息请参阅GDBStub。

紧急处理程序的行为还受到另外两个配置项的影响：

- 如果使能了CONFIG_ESP_DEBUG_OCDWARE（默认），紧急处理程序会检测ESP32是否已经连接JTAG调试器。如果检测成功，程序会暂停运行，并将控制权交给调试器。在这种情况下，寄存器和回溯不会被打印到控制台，并且也不会使用GDB Stub和Core Dump的功能。
- 如果使能了内核转储功能，系统状态（任务堆栈和寄存器）会被转储到flash或者UART以供后续分析。
- 如果CONFIG_ESP_PANIC_HANDLER_IRAM被禁用（默认情况下禁用），紧急处理程序的代码会放置在flash而不是IRAM中。这意味着，如果ESP-IDF在flash高速缓存禁用时崩溃，在运行GDB Stub和内核转储之前紧急处理程序会自动重新使能flash高速缓存。如果flash高速缓存也崩溃了，这样做会增加一些小风险。
- 如果使能了该选项，紧急处理程序的代码（包括所有的UART函数）会放置在IRAM中，导致SRAM中的可用内存空间变小。当禁用flash高速缓存（如写入SPI flash时）或触发异常导致flash高速缓存崩溃时，可用此选项调试一些复杂的崩溃问题。

下图展示了紧急处理程序的行为：

### 4.13.3 寄存器转储与回溯

除非启用了CONFIG_ESP_SYSTEM_PANIC_SILENT_REBOOT否则紧急处理程序会将CPU寄存器和回溯打印到控制台

<table>
<thead>
<tr>
<th>Core 0 register dump:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
</tr>
<tr>
<td>A1</td>
</tr>
<tr>
<td>A2</td>
</tr>
<tr>
<td>A3</td>
</tr>
<tr>
<td>A4</td>
</tr>
<tr>
<td>A5</td>
</tr>
<tr>
<td>A6</td>
</tr>
<tr>
<td>A7</td>
</tr>
<tr>
<td>A8</td>
</tr>
<tr>
<td>A9</td>
</tr>
<tr>
<td>A10</td>
</tr>
<tr>
<td>A11</td>
</tr>
<tr>
<td>A12</td>
</tr>
<tr>
<td>A13</td>
</tr>
<tr>
<td>A14</td>
</tr>
<tr>
<td>A15</td>
</tr>
<tr>
<td>A16</td>
</tr>
<tr>
<td>A17</td>
</tr>
<tr>
<td>A18</td>
</tr>
<tr>
<td>A19</td>
</tr>
<tr>
<td>A20</td>
</tr>
<tr>
<td>A21</td>
</tr>
<tr>
<td>A22</td>
</tr>
<tr>
<td>A23</td>
</tr>
<tr>
<td>A24</td>
</tr>
<tr>
<td>A25</td>
</tr>
<tr>
<td>A26</td>
</tr>
<tr>
<td>A27</td>
</tr>
<tr>
<td>A28</td>
</tr>
<tr>
<td>A29</td>
</tr>
<tr>
<td>A30</td>
</tr>
<tr>
<td>A31</td>
</tr>
<tr>
<td>EXCVADR:</td>
</tr>
<tr>
<td>LBEG:</td>
</tr>
<tr>
<td>LEND:</td>
</tr>
<tr>
<td>LCOUNT:</td>
</tr>
<tr>
<td>Backtrace:</td>
</tr>
</tbody>
</table>

仅会打印异常帧中CPU寄存器的值，即引发CPU异常或者其它严重错误时刻的值。
紧急处理程序如果因abort()而调用，则不会打印寄存器转储。
图 34: 紧急处理程序流程图（点击放大）
在某些情况下，例如中断看门狗超时，紧急处理程序会额外打印 CPU 寄存器（EPC1-EPC4）的值，以及另一个 CPU 的寄存器值和代码回溯。

回溯行包含了当前任务中每个堆栈帧的 PC:SP 对（PC 是程序计数器，SP 是堆栈指针）。如果在 ISR 中发生了严重错误，回溯会同时包括被中断任务的 PC:SP 对，以及 ISR 中的 PC:SP 对。

如果使用了 IDF 监视器，该工具会将程序计数器的值转换为对应的代码位置（函数名，文件名，行号），并加以注释。

<table>
<thead>
<tr>
<th>Core 0 register dump:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC : 0x400e14ed</td>
</tr>
<tr>
<td>A2 : 0x00000000</td>
</tr>
<tr>
<td>A6 : 0x00000000</td>
</tr>
<tr>
<td>A10 : 0x00000000</td>
</tr>
<tr>
<td>A14 : 0x3ffb7078</td>
</tr>
</tbody>
</table>

若要查找发生严重错误的代码位置，请查看“Backtrace”的后面几行，发生严重错误的代码显示在顶行，后续几行显示的是调用堆栈。

### 4.13.4 GDB Stub

如果启用了 CONFIG_ESP_SYSTEM_PANIC_GDBSTUB 选项，在发生严重错误时，紧急处理程序不会复位芯片，相反，它将启动 GDB 远程协议服务器，通常称为 GDB Stub。发生这种情况时，可以让主机上运行的 GDB 实例通过 UART 端口连接到 ESP32。

如果使用了 IDF 监视器，该工具会在 UART 端口检测到 GDB Stub 提示符后自动启动 GDB，输出会类似于：

```
Entering gdb stub now.
$70b#6GNU gdb (crosstool-NG crosstool-ng-1.22.0-80-gff1f415) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-build_apple-darwin16.3.0 --target=xtensa-
esp32-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"... Reading symbols from /Users/user/esp/example/build/example.elf...done.
Remote debugging using /dev/cu.usbserial-31301
```
在 GDB 会话中，我们可以检查 CPU 寄存器，本地和静态变量以及内存中任意位置的值。但是不支持设置断点，改变 PC 值或者恢复程序的运行。若要复位程序，请退出 GDB 会话，在 IDF 监视器中连续输入 Ctrl-T Ctrl-R，或者按下开发板上的复位按键也可以重新运行程序。

### 4.13.5 RTC 看门狗超时

RTC 看门狗在启动代码中用于跟踪执行时间，也有助于防止由于电源不稳定引起的锁死。RTC 看门狗默认启用，参见 `CONFIG_BOOTLOADER_WDT_ENABLE`。如果执行时间超时，RTC 看门狗将自动重启系统。此时，ROM 引导加载程序将打印消息 RTC Watchdog Timeout 说明重启原因。

<table>
<thead>
<tr>
<th>rst:0x10 (RTCWDT_RTC_RESET)</th>
</tr>
</thead>
</table>

RTC 看门狗涵盖了从一级引导程序 (ROM 引导程序) 到应用程序启动的执行时间，最初在 ROM 引导程序中设置。在后台引导程序中使用 `CONFIG_BOOTLOADER_WDT_TIME_MS` 选项进行配置 (默认 9000 ms)。在应用初始化阶段，由于慢速时钟源可能已更改，RTC 看门狗将被重新配置，最后在调用 `app_main()`之前被禁用。可以使用选项 `CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE` 以保证 RTC 看门狗在调用 `app_main()` 之前不被禁用，而是保持运行状态，用户需要在应用代码中定期 “喂狗”。

### 4.13.6 Guru Meditation 错误

本节将对打印在 Guru Meditation Error: Core panic’ed 后面括号中的致错原因进行逐一解释。

**备注：** 想要了解 “Guru Meditation”的历史渊源，请参阅 维基百科。

**IllegalInstruction**

此 CPU 异常表示当前执行的指令不是有效指令，引起此错误的常见原因包括：

- FreeRTOS 中的任务函数已返回。在 FreeRTOS 中，如果想终止任务函数，需要调用 `vTaskDelete()` 函数释放当前任务的资源，而不是直接返回。
- 无法从 SPI flash 中读取下一条指令，这通常发生在：
  - 应用程序将 SPI flash 的管脚重新配置为其它功能 (如 GPIO、UART 等)。有关 SPI flash 管脚的详细信息，请参阅硬件设计指南和芯片/模组的数据手册。
  - 某些外部设备意外连接到 SPI flash 的管脚上，干扰了 ESP32 和 SPI flash 之间的通信。
- 在 C++ 代码中，退出 non-void 函数而无返回值被认为是未定义的行为。启用后编译器通常会忽略此类函数的结尾，导致 IllegalInstruction 异常。默认情况下，ESP-IDF 构建系统启用 `-Werror=return-type`，这意味着缺少返回语句会被视为编译时错误。但是，如果应用程序项目禁用了编译器警告，可能就无法检测到该问题，在运行时就会出现 IllegalInstruction 异常。

**InstrFetchProhibited**

此 CPU 异常表示 CPU 无法读取指令，因为指令的地址不在 IRAM 或者 IROM 中的有效区域中。

通常这意味着代码中调用了并不指向有效代码块的函数针。这种情况中，可以查看 PC (程序计数器) 寄存器的值并做出进一步判断：若为 0 或者其它非法值 (即只要不是 0x4xxxxxxx 的情况)，则证实确实是该原因。
LoadProhibited, StoreProhibited

当应用程序尝试读取或写入无效的内存位置时，会发生此类 CPU 异常。此类无效内存地址可以在寄存器转储的 EXCVADDR 中找到。如果该地址为零，通常意味着应用程序尝试访问一个 NULL 指针。如果该地址接近于零，则通常意味着应用程序尝试访问某个结构体的成员，但是该结构体的指针为 NULL。如果该地址是其它非法值（不在 0x3fxxxxxx - 0x6xxxxxxx 的范围内），则可能意味着用于访问数据的指针未初始化或者已经损坏。

IntegerDivideByZero

应用程序尝试将整数除以零。

LoadStoreAlignment

应用程序尝试读取/写入的内存位置不符合加载/存储指令对字节对齐大小的要求。例如，32 位读取指令只能访问 4 字节对齐的内存地址，而 16 位写入指令只能访问 2 字节对齐的内存地址。

LoadStoreError

这类异常通常发生于以下几种场合:

- 应用程序尝试从仅支持 32 位读取/写入的内存区域执行 8 位或 16 位加载/存储操作，例如，解引用一个指向指令内存区域（比如 IRAM 或者 IROM）的 char* 指针就会触发这个错误。
- 应用程序尝试写入数据到只读的内存区域（比如 IROM 或者 DROM）也会触发这个错误。

Unhandled debug exception

这后面通常会再跟一条消息:

```
Debug exception reason: Stack canary watchpoint triggered (task_name)
```

此错误表示应用程序写入的位置越过了 task_name 任务堆栈的末尾，请注意，并非每次堆栈溢出都会触发此错误。任务有可能会绕过堆栈栈底（stack canary）的位置访问内存，在这种情况下，监视点就不会被触发。

Interrupt wdt timeout on CPU0 / CPU1

这表示发生了中断看门狗超时，详细信息请查阅看门狗 文档。

Cache disabled but cached memory region accessed

在某些情况下，ESP-IDF 会暂时禁止通过高速缓存访问外部SPI flash 和 SPI RAM，例如在使用 spi_flash API 读取/写入/擦除/映射 SPI flash 的时候。在这些情况下，任务会被挂起，并且未使用 ESP_INTR_FLAG_IRAM 注册的中断处理程序会被禁用。请确保任何使用此标志注册的中断处理程序所访问的代码和数据分别位于 IRAM 和 DRAM 中。更多详细信息请参阅SPI flash API 文档。

4.13.7 其他严重错误

掉电

ESP32 内部集成掉电检测电路，并且会默认启用。如果电源电压低于安全值，掉电检测器可以触发系统复位。掉电检测器可以使用 CONFIG_ESP_BROWNOUT_DET 和 CONFIG_ESP_BROWNOUT_DET_LVL_SEL 这两个选项进行设置。
当电网检测器被触发时，会打印如下信息：

Brownout detector was triggered

芯片会在该打印信息结束后复位。
请注意，如果电源电压快速下降，则只能在控制台上看到部分打印信息。

堆不完整

ESP-IDF 堆的实现包含许多运行时的堆结构检查，可以在 menuconfig 中开启额外的检查（“Heap Poisoning”）。如果其中的某项检查失败，则会打印类似如下信息：

```
CORRUPT HEAP: Bad tail at 0x3ffe270a. Expected 0xbaad5678 got 0xbaac5678
assertion "head != NULL" failed: file "/Users/user/esp/esp-idf/components/heap/multi_heap_poisoning.c", line 201, function: multi_heap_free
abort() was called at PC 0x400dca43 on core 0
```

更多详细信息，请查阅堆内存调试文档。

堆栈粉碎

堆栈粉碎保护（基于 GCC -fstack-protector* 标志）可以通过 ESP-IDF 中的 CONFIG_COMPILER_STACK_CHECK_MODE 选项来开启。如果检测到堆栈粉碎，则会打印类似如下的信息：

```
Stack smashing protect failure!
abort() was called at PC 0x400d2138 on core 0
Backtrace: 0x4008e6c0:0x3ffc1780 0x4008e8b7:0x3ffc17a0 0x400d2138:0x3ffc17c0...
  →0x400e79d5:0x3ffc17e0 0x400e79a7:0x3ffc1840 0x400e79df:0x3ffc18a0...
  →0x400e2235:0x3ffc18c0 0x400e1916:0x3ffc18f0 0x400e19cd:0x3ffc1910...
  →0x400e1a11:0x3ffc1930 0x400e1bb2:0x3ffc1950 0x400d2c44:0x3ffc1a80
```

回溯信息会指明发生堆栈粉碎的函数，建议检查函数中是否有代码访问局部数组时发生了越界。

未定义行为清理器 (UBSAN) 检查

未定义行为清理器 (UBSAN) 是一种编译器功能，它会为可能不正确的操作添加运行时检查，例如：

- 溢出（乘法溢出、有符号整数溢出）
- 移位基数或指数错误（如移位超过 32 位）
- 整数转换错误

请参考 GCC 文档中的 "-fsanitize=undefined" 选项，查看支持检查的完整列表。

使用 UBSAN 默认情况下未使用 UBSAN。可以通过在构建系统中添加编译器选项
- fsanitize=undefined 在文件、组件或项目级别上使用 UBSAN。

在对使用 SoC 硬件寄存器头文件（soc/xxx_reg.h）的代码使用 UBSAN 时，建议使用
- fno-sanitize=shift-base 选项禁用移位基数清理器。这是由于 ESP-IDF 寄存器头文件目前包含的模式会对这个特定的清理器选项造成误报。

要在项目级别使能 UBSAN，请在项目 CMakeLists.txt 文件的末尾添加以下内容：

```
idf_build_set_property(COMPILE_OPTIONS "-fsanitize=undefined" "-fno-sanitize=shift-base" APPEND)
```
或者，通过 EXTRA_CFLAGS 和 EXTRA_CXXFLAGS 环境变量来传递这些选项。
使能 UBSAN 会明显增加代码量和数据大小。当为整个应用程序使能 UBSAN 时，微控制器的可用 RAM 无法容纳大多数应用程序（除了一些微小程序）。因此，建议为特定的待测组件使能 UBSAN。
要为项目 CMakeLists.txt 文件中的特定组件（component_name）启用 UBSAN，请在文件末尾添加以下内容:

```cmake
idf_component_get_property(lib component_name COMPONENT_LIB)
target_compile_options(${lib} PRIVATE "-fsanitize=undefined" "-fno-sanitize=shift-base")
```

注意：关于构建属性 和 组件属性 的更多信息，请查看构建系统文档。

要为同一组件的 CMakeLists.txt 中的特定组件（component_name）使能 UBSAN，在文件末尾添加以下内容:

```cmake
target_compile_options(${COMPONENT_LIB} PRIVATE "-fsanitize=undefined" "-fno-sanitize=shift-base")
```

**UBSAN 输出** 当 UBSAN 检测到一个错误时，会打印一个信息和回溯，例如:

```
Undefined behavior of type out_of_bounds
Backtrace: 0x4008b383:0x3ffcd8b0 0x4008c791:0x3ffcd8d0 0x4008c587:0x3ffcd8f0... 0x4008c6be:0x3ffcd950 0x400db74f:0x3ffcd970 0x400db99c:0x3ffcd9a0
```

当使用 *IFD 监视器* 时，回溯会被解码为函数名以及源代码位置，并指向问题发生的位置（这里是 main.c:128）:

```
0x4008b383: panic_abort at /path/to/esp-idf/components/esp_system/panic.c:367
0x4008c791: esp_system_abort at /path/to/esp-idf/components/esp_system/system_api. →c:106
0x4008c587: __ubsan_default_handler at /path/to/esp-idf/components/esp_system/ →ubsan.c:152
0x4008c6be: __ubsan_handle_out_of_bounds at /path/to/esp-idf/components/esp_system/ →ubsan.c:223
0x400db74f: test_ub at main.c:128
0x400db99c: app_main at main.c:56 (discriminator 1)
```

UBSAN 报告的错误类型为以下几种:
4.14 Flash 加密

本文档旨在引导用户快速了解 ESP32 的 flash 加密功能，通过应用程序代码示例向用户演示如何在开发及生产过程中测试及验证 flash 加密的相关操作。

4.14.1 概述

flash 加密功能用于加密与 ESP32 搭载使用的片外 flash 中的内容。启用 flash 加密功能后，固件会以明文形式烧录，然后在首次启动时将数据进行加密。因此，物理读取 flash 将无法恢复大部分 flash 内容。

启用 flash 加密后，系统将默认加密下列类型的 flash 数据:

- 固件引导加载程序
- 分区表
- 所有“app”类型的分区

其他类型的数组将视情况加密处理:

- 任何在分区表中标有“加密”标志的分区。详情请见加密分区标志。
- 如果启用了安全启动，则可以加密安全启动引导程序摘要（见下文）。

安全启动 是一个独立的功能，可以与 flash 加密一起使用，从而创建更安全的环境。

重要: 对于生产用途，flash 加密仅应在“发布”模式下启用。

重要: 启用 flash 加密将限制后续 ESP32 更新。在使用 flash 加密功能前，请务必阅读本文档了解其影响。

4.14.2 相关 eFuses

Flash 加密操作由 ESP32 上的多个 eFuse 控制。以下是这些 eFuse 列表及其描述。下表中的各 eFuse 名称也在 espefuse.py 工具中使用，为了能在 eFuse API 中使用，而需在名称前加上 ESP_EFuse 模块，如: esp_efuse_read_field_bit(ESP_EFUSE_DISABLE_DL_ENCRYPT)。
表 20: Flash 加密过程中使用的 eFuses

<table>
<thead>
<tr>
<th>eFuse</th>
<th>描述</th>
<th>位深</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODING_SCHEMA</td>
<td>控制用于产生最终 256 位 AES 密钥的 block1 的实际位数。可能的值：0 代表 256 位，1 代表 192 位，2 代表 128 位。最终的 AES 密钥根据 FLASH_CRYPT_CONFIG 值得出。</td>
<td>2</td>
</tr>
<tr>
<td>flash_encryption (block1)</td>
<td>AES 密钥存储。</td>
<td>256 位密钥块</td>
</tr>
<tr>
<td>FLASH_CRYPT_CONFIG</td>
<td>控制 AES 加密过程。</td>
<td>4</td>
</tr>
<tr>
<td>DISABLE_DL_ENCRYPT</td>
<td>设置后，在固件下载模式运行时禁用 flash 加密操作。</td>
<td>1</td>
</tr>
<tr>
<td>DISABLE_DL_DECRYPT</td>
<td>设置后，在 UART 固件下载模式运行时禁用 flash 解密操作。</td>
<td>1</td>
</tr>
<tr>
<td>FLASH_CRYPT_CNT</td>
<td>在启动时启用/禁用加密。如果设置了偶数个比特位 (0, 2, 4, 6), 则在启动时加密 flash。如果设置了奇数个比特位 (1, 3, 5, 7), 则在启动时不加密 flash。</td>
<td>7</td>
</tr>
</tbody>
</table>

备注:
- 上表中列出的所有 eFuse 位都提供读/写访问控制。
- 这些位的默认值是 0。

对上述 eFuse 位的读写访问由 WR_DIS 和 RD_DIS 寄存器中的相应字段控制。有关 ESP32 eFuse 的详细信息，请参考 eFuse 管理器。要使用 espefuse.py 更改 eFuse 字段的保护位，请使用以下两个命令: read_protect_efuse 和 write_protect_efuse。例如 espefuse.py write_protect_efuse DISABLE_DL_ENCRYPT。

4.14.3 Flash 的加密过程

假设 eFuse 值处于默认状态，且固件的引导加载程序编译为支持 flash 加密，则 flash 加密的具体过程如下:

1. 第一次开机复位时，flash 中的所有数据都是未加密的 (明文)。ROM 引导加载程序加载固件引导加载程序。
2. 固件的引导加载程序将读取 FLASH_CRYPT_CNT eFuse 值 (0b00000000)。因为该值为 0 (偶数位)，固件的引导加载程序将配置并启用 flash 加密块，同时将 FLASH_CRYPT_CONFIG eFuse 的值编程为 0xF。关于 flash 加密块的更多信息，请参考 ESP32 技术参考手册 > eFuse 控制器 (eFuse) > flash 加密块 [PDF]。
3. 固件的引导加载程序使用 RNG（随机数生成）模块生成 AES-256 位密钥，然后将其写入 flash_encryption eFuse 中。由于 flash_encryption eFuse 已设置密码和读/写保护位，将无法通过软件访问密钥。Flash 加密操作完全在硬件中完成，无法通过软件访问密钥。
4. Flash 加密块将加密 flash 的内容（固件的引导加载程序、应用程序以及标有“加密”标志的分区）。加密可能需要一些时间（对于大分区最多需要一分钟）。
5. 固件引导加载程序将在 FLASH_CRYPT_CNT (0b00000001) 中设置第一个可用位来对已加密的 flash 内容进行标记。设置奇数个比特位。
6. 对于开发模式，固件引导加载程序仅设置 DISABLE_DL_DECRYPT 和 DISABLE_DL_CACHE 的 eFuse 位，以便 UART 引导加载程序重新编译加密的二进制文件。此外，FLASH_CRYPT_CNT 的 eFuse 位不受写入保护。
7. 对于发布模式，固件引导加载程序设置 DISABLE_DL_ENCRYPT、DISABLE_DL_DECRYPT 和 DISABLE_DL_CACHE 的 eFuse 位为 1，以防止 UART 引导加载程序解密 flash 内容。它还写保护 FLASH_CRYPT_CNT eFuse 位。要修改此行为，请参阅启用 UART 引导加载程序加密/解密。
8. 重新启动设备以开始执行加密镜像。固件引导加载程序调用 flash 解密块来解密 flash 内容，然后将解密的内容加载到 IRAM 中。

在开发阶段常需编写不同的明文 flash 镜像并测试 flash 的加密过程。这要求固件下载模式能够根据需求不断加载新的明文镜像。但即使在制造和生产过程中，出于安全考虑，固件下载模式不应有权限访问 flash 内容。
因此需要有两种不同的 flash 加密配置：一种用于开发，另一种用于生产。详情请参考 Flash 加密设置 小节。

### 4.14.4 Flash 加密设置

提供以下 flash 加密模式：

- **开发模式** - 建议仅在开发过程中使用。因为在这种模式下，仍然可以将新的明文固件烧录到设备，并且引导加载程序将使用存储在硬件中的密钥对该固件进行透明加密。此操作间接允许从 flash 中读出固件密文。
- **发布模式** - 推荐用于制造和生产。因为在这种模式下，如果不知道加密密钥，则不可能将明文固件烧录到设备。

本节将详细介绍上述 flash 加密模式，并且逐步说明如何使用它们。

#### 开发模式

在开发过程中，可使用 ESP32 内部生成的密钥或外部主机生成的密钥进行 flash 加密。

**使用 ESP32 生成的密钥** 开发模式允许用户使用固件下载模式下载多个明文镜像。

测试 flash 加密过程需完成以下步骤：

1. 确保您的 ESP32 设备有相关 eFuses 中所示的 flash 加密 eFuse 的默认设置。
   请参考如何检查 ESP32 flash 加密状态。
2. 在项目配置菜单，执行以下操作：
   - 启动时使能 flash 加密。
   - 选择加密模式 （默认是 开发模式）。
   - 选择 UART ROM 下载模式 （默认是 启用）。请注意，对于 ESP32 芯片，该选项仅在 `CONFIG_ESP32_REV_MIN` 级别设置为 3 时 (ESP32 V3) 可用。
   - 选择适当详细程度的引导加载程序日志。
   - 保存配置并退出。

启用 flash 加密将增大引导加载程序，因而可能需更新分区表偏移量。请参考引导加载程序大小。

3. 运行以下命令来构建和烧录完整的镜像。

   ```
   idf.py flash monitor
   ```

**备注**：这个命令不包括任何应该写入 flash 分区的用户文件。请在运行此命令前手动写入这些文件，否则在写入前应单独对这些文件进行加密。

该命令将向 flash 写入未加密的镜像：固件引导加载程序、分区表和应用程序。烧录完成后，ESP32 将复位。在下一次启动时，固件引导加载程序会加密：固件引导加载程序、应用程序分区和标记为“加密”的分区。然后复位。就加密可能需要时间，对于大分区最多需要一分钟。之后，应用程序在运行时解密并执行命令。

下面是启用 flash 加密后 ESP32 首次启动时的样例输出：

```
--- idf_monitor on /dev/cu.SLAB_USBtoUART 115200 ---
--- Quit: Ctrl+| | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun 8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configskip: 0, SPIWP:0xee
```
### Chapter 4. API 指南

| click_drv: 0x00, q_drv: 0x00, d_drv: 0x00, cs0_drv: 0x00, hd_drv: 0x00, wp_drv: 0x00 | mode: DIO, clock div: 2 |
| load: 0x3fff0018, len: 4 |
| load: 0x40078000, len: 13608 |
| load: 0x40080400, len: 664 |
| entry 0x40080764 |
| I (28) boot: ESP-IDF V4.0-dev-850-gc4447462d-dirty 2nd stage bootloader |
| I (29) boot: compile time: 15:37:14 |
| I (30) boot: Enabling RNG early entropy source... |
| I (35) boot: SPI Speed: 40MHz |
| I (39) boot: SPI Mode: DIO |
| I (43) boot: SPI Flash Size: 4MB |
| I (47) boot: Partition Table: |
| I (51) boot: Entry 0x40080764 at esp-idf/esp-idf/components/freertos/xtensa_vectors. |
| I (73) boot: factory app at offset 0x20000 |
| I (81) boot: End of partition table |
| I (85) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x0808c (仿佛32908) map |
| I (104) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3fff0000 size=0x01e4a (仿佛7844) load |
| I (107) esp_image: segment 2: paddr=0x00029f60 vaddr=0x40080000 size=0x00400 (仿佛1024) load |
| 0x40080000: _WindowOverflow4 at esp-idf/esp-idf/components/freertos/xtensa_vectors. |
| I (114) esp_image: segment 3: paddr=0x0002a368 vaddr=0x40080400 size=0x05ca8 (仿佛23720) load |
| I (121) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018 size=0x126a8 (仿佛75432) map |
| 0x400d0018: _flash_cache_start at ???: |
| I (159) esp_image: segment 5: paddr=0x000426c8 vaddr=0x400860a8 size=0x01f4c (仿佛8012) load |
| 0x400860a8: prvAddNewTaskToReadyList at esp-idf/esp-idf/components/freertos/tasks. |
| I (168) boot: Loaded app from partition at offset 0x20000 |
| I (168) boot: Checking flash encryption... |
| I (168) flash_encrypt: Generating new flash encryption key... |
| I (177) flash_encrypt: Read & write protecting new key... |
| I (187) flash_encrypt: Setting CRYPT_CONFIG efuse to 0xF |
| W (188) flash_encrypt: Not disabling UART bootloader encryption |
| I (195) flash_encrypt: Disable UART bootloader decryption... |
| I (201) flash_encrypt: Disable UART bootloader MMU cache... |
| I (208) flash_encrypt: Disable JTAG... |
| I (212) flash_encrypt: Disable ROM BASIC interpreter fallback... |
| I (219) esp_image: segment 0: paddr=0x00001020 vaddr=0x3fff0018 size=0x00004 (仿佛4) |
| I (227) esp_image: segment 1: paddr=0x0000102c vaddr=0x3fff001c size=0x02104 (仿佛8452) |
| I (239) esp_image: segment 2: paddr=0x00003138 vaddr=0x40078000 size=0x03528 (仿佛13608) |
| I (249) esp_image: segment 3: paddr=0x00006668 vaddr=0x40080400 size=0x01a08 (仿佛6664) |
| I (657) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x000008c (仿佛32908) map |
| I (669) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3fff0000 size=0x01e4a (仿佛7844) |

(续上页)
Chapter 4. API 指南

(续上页)

I (672) esp_image: segment 2: paddr=0x00029f60 vaddr=0x40080000 size=0x00400 ( _-
~1024)
0x40080000: _WindowOverflow4 at esp-idf/esp-idf/components/freertos/xtensa_vectors.
~S:1778

I (676) esp_image: segment 3: paddr=0x0002a368 vaddr=0x40080400 size=0x05ca8 ( _-
~23720)
I (692) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018 size=0x126a8 ( _-
~75432) map
0x400d0018: _flash_cache_start at ???:?

I (719) esp_image: segment 5: paddr=0x000426c8 vaddr=0x400860a8 size=0x01f4c ( _-
~8012)
0x400860a8: prvAddNewTaskToReadyList at esp-idf/esp-idf/components/freertos/tasks.
~c:4561

I (722) flash_encrypt: Encrypting partition 2 at offset 0x20000...
I (13229) flash_encrypt: Flash encryption completed
I (13229) boot: Resetting with flash encryption enabled...

启用 flash 加密后，在下次启动时输出将显示已启用 flash 加密，样例输出如下：

rst:0x1 (POWERON_RESET), boot:0x13 (SPI_FAST_FLASH_BOOT)
configspi: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8452
load:0x40078000,len:13652
load:0x40080400,len:6664
entry 0x40080764

I (30) boot: ESP-IDF v4.0-dev-850-gc4447462d-diary 2nd stage bootloader
I (30) boot: compile time 16:32:53
I (31) boot: Enabling RNG early entropy source...
I (37) boot: SPI Speed : 40MHz
I (41) boot: SPI Mode : DIO
I (45) boot: SPI Flash Size : 4MB
I (49) boot: Partition Table:
I (52) boot: ## Label           Usage          Type ST Offset Length
I (60) boot: 0 nvs            WiFi data      01 02 0000a000 00006000
I (67) boot: 1 phy_init        RF data        01 01 00010000 00010000
I (75) boot: 2 factory        factory app   00 00 00020000 00100000
I (82) boot: End of partition table
I (86) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x0808c (~
~32908) map
I (107) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3ff0000 size=0x01ea4 ( _-
~7844) load
I (111) esp_image: segment 2: paddr=0x00029f60 vaddr=0x40080000 size=0x00400 ( _-
~1024) load
0x40080000: _WindowOverflow4 at esp-idf/esp-idf/components/freertos/xtensa_-
vectors.S:1778

I (116) esp_image: segment 3: paddr=0x0002a368 vaddr=0x40080400 size=0x05ca8 ( _-
~23720) load
I (134) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018 size=0x126a8 ( _-
~75432) map
0x400d0018: _flash_cache_start at ???:?
I (162) esp_image: segment 5: paddr=0x000426c8 vaddr=0x400860a8 size=0x01f4c ( _-
~8012) load

(下页继续)
使用主机生成的密钥  可在主机中预生成 flash 加密密钥，并将其烧录到 eFuse 密钥烧录块中。这样，无需明文 flash 更新便可以在主机上预加密数据并将其烧录。该功能可在开发模式 和发布模式 两模式下使用。如果没有预生成的密钥，数据将使用 明文方式烧录，然后 ESP32 对数据进行就地加密。

备注：不建议在生产中使用该方法，除非为每个设备单独生成一个密钥。

使用主机生成的密钥需完成以下步骤：

1. 确保您的 ESP32 设备有相关 eFuses 中所示的 flash 加密 eFuse 的默认设置。
   请参考如何检查 ESP32 flash 加密状态。

2. 通过运行以下命令生成一个随机密钥：
   ```
   espsecure.py generate_flash_encryption_key my_flash_encryption_key.bin
   ```

3. 在第一次加密启动前，使用以下命令将该密钥烧录到设备上，这个操作只能执行 一次。
   ```
   espefuse.py --port PORT burn_key flash_encryption my_flash_encryption_key.bin
   ```

| 0x400860a8: prvAddNewTaskToReadyList at esp-idf/esp-idf/components/freertos/tasks.c:4561 |
|---|---|
| I (171) boot: Loaded app from partition at offset 0x20000 |
| I (171) boot: Checking flash encryption... |
| I (171) flash_encrypt: flash encryption is enabled (3 plaintext flashes left) |
| I (178) boot: Disabling RNG early entropy source... |
| I (184) cpu_start: Pro cpu up. |
| I (188) cpu_start: Application information: |
| I (193) cpu_start: Project name: flash-encryption |
| I (198) cpu_start: App version: v4.0-dev-850-gc4447462d-dirty |
| I (205) cpu_start: Compile time: Jun 17 2019 16:32:52 |
| I (211) cpu_start: ELF file SHA256: 8770c886bdf561a7... |
| I (217) cpu_start: ESP-IDF: v4.0-dev-850-gc4447462d-dirty |
| I (224) cpu_start: Starting app cpu, entry point is 0x40080e4c |
| 0x40080e4c: call_start_cpu1 at esp-idf/esp-idf/components/esp32/cpu_start.c:265 |
| I (0) cpu_start: App cpu up. |
| I (235) heap_init: Initializing. RAM available for dynamic allocation: |
| I (241) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM |
| I (247) heap_init: At 3FFB2EC8 len 0002D138 (180 KiB): DRAM |
| I (254) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM |
| I (260) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM |
| I (266) heap_init: At 40087FF4 len 0001800C (96 KiB): IRAM |
| I (273) cpu_start: Pro cpu start user code |
| I (291) cpu_start: Starting scheduler on PRO CPU. |
| I (0) cpu_start: Starting scheduler on APP CPU. |

Sample program to check Flash Encryption
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 4MB external flash
Flash encryption feature is enabled
Flash encryption mode is DEVELOPMENT
Flash in encrypted mode with flash_encrypt_cnt = 1
Halting...
如果未烧录密钥并在启动 flash 加密后启动设备，ESP32 将生成一个软件无法访问或修改的随机密钥。

4. 在项目配置菜单中进行如下设置:
   - 启动时启用 flash 加密功能
   - 选择加密模式（默认为开发模式）
   - 选择适当详细程度的引导加载程序日志
   - 保存配置并退出

启用 flash 加密将增大引导加载程序，因而可能需更新分区表偏移量。请参考引导加载程序大小。

5. 运行以下命令来构建并烧录完整的镜像:

   ```sh
   idf.py flash monitor
   ```

**备注:** 这个命令不包括任何应该被写入 flash 上的分区的用户文件。请在运行此命令前手动写入这些文件，否则在写入前应单独对这些文件进行加密。

该指令将向 flash 写入未加密的镜像：固件引导加载程序、分区表和应用程序。烧录完成后，ESP32 将复位。在下一次启动时，固件引导加载程序会加密：固件引导加载程序、应用程序分区和标记为加密的分区，然后复位。就地加密可能需要时间，对于大的分区来说可能耗时一分钟。之后，应用程序在运行时被解密并执行。

如果使用开发模式，那么更新和重新烧录二进制文件最简单的方法是重新烧录更新后的分区。

如果使用发布模式，那么可以在主机上预先加密二进制文件，然后将其作为密文烧录。具体请参考手动加密文件。

**重新烧录更新后的分区** 如果用户用明文方式更新了应用程序代码并需要重新烧录，则需要在烧录前对其进行加密。请运行以下命令一次完成应用程序的加密与烧录：

```sh
idf.py encrypted-app-flash monitor
```

如果所有分区都需要加密形式更新，请运行：

```sh
idf.py encrypted-flash monitor
```

**发布模式**

在发布模式下，UART 引导加载程序无法执行 flash 加密操作。只能使用 OTA 方案下载新的明文镜像，该方案将在写入 flash 前加密明文镜像。

使用该模式需要执行以下步骤：

1. 确保您的 ESP32 设备有相关 eFuses 中所示的 flash 加密 eFuse 的默认设置。
   - 请参考如何检查 ESP32 flash 加密状态。
2. 在项目配置菜单，执行以下操作：

   - 启动时便能 flash 加密
   - 选择发布模式（注意一旦选择了发布模式，DISABLE_DL_ENCRYPT 和 DISABLE_DL_DECRYPT eFuse 位将被编程为在 ROM 下载模式下禁用 flash 加密硬件）
   - 选择 UART ROM 下载模式（推荐永久性禁用）（注意该选项仅在 CONFIG_ESP32_REV_MIN 级别设置为 3 时（ESP32 V3 可用。）默认选项是保持启用 UART ROM 下载模式，然而建议永久禁用该模式，以减少攻击者可用的选项。
   - 选择适当详细程度的引导加载程序日志
   - 保存配置并退出
启用 flash 加密将增大引导加载程序，因而可能需更新分区表偏移量。请参考引导加载程序大小。

3. 运行以下命令来构建并烧录完整的镜像:

```
idf.py flash monitor
```

**备注:** 这个命令不包括任何应该被写入 flash 区域的用户文件。请在运行此命令前手动写入这些文件，否则在写入前应单独对这些文件进行加密。

该命令将向 flash 写入未加密的镜像：固件引导加载程序、分区表和应用程序。烧录完成后，ESP32 将复位。在下一次启动时，固件引导加载程序会加密：固件引导加载程序、应用程序分区和标记为 加密的分区，然后复位。就地加密可能需要时间，对于大的分区来说可能耗时一分钟。之后，应用程序在运行时被解密并执行。

一旦在发布模式下启用 flash 加密，引导加载程序将写保护 FLASH_CRYPT_CNT eFuse。

请使用OTA 方案对字段中的明文进行后续更新。

**备注:** 如果用户已经预先生成了 flash 加密密钥并存储了一个副本，并且 UART 下载模式没有通过 CONFIG_SECURE_UART_ROM_DL_MODE (ESP32 V3 only) 永久禁用，那么可以通过使用 espsecure.py encrypt_flash_data 预加密文件，从而在本地更新 flash，然后烧录密文。请参看手动加密文件。  

**最佳实践**

在生产中使用 flash 加密时：

- 不要在多个设备之间重复使用同一个 flash 加密密钥，这样攻击者就无法从一台设备上复制加密数 据再将其转移到第二台设备上。
- 在使用 ESP32 V3 时，如果生产设备需要 UART ROM 下载模式，那么则该禁用该模式以增加设 备安全性。这可以通过在应用程序启动时调用esp_efuse_disable_rom_download_mode() 来实现。或者，可将项目 CONFIG_ESP32_REV_MIN 级别配置为 3 (仅针对 ESP32 V3)，然后选 择 CONFIG_SECURE_UART_ROM_DL_MODE 为“永久性的禁用 ROM 下载模式 (推荐)”。在早期的 ESP32 版本上无法禁用 ROM 下载模式。
- 启用安全启动 作为额外的保护层，防止攻击者在启动前有选择地破坏 flash 中某部分。

### 4.14.5 可能出现的错误

一旦启用 flash 加密，FLASH_CRYPT_CNT 的 eFuse 值将设置为奇数值。这意味着所有标有加密标志的分 区都会包含加密的密文。如果 ESP32 错误地加载了明文数据，则会出现以下三种典型的错误情况：

1. 如果通过 明文固件引导加载程序镜像重新烧录了引导加载程序分区，则 ROM 加载器将无法加载 固件引导加载程序，并会显示以下错误类型:

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57
```

```
rst:0x7 (TG0WD_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57
```

```
rst:0x7 (TG0WD_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
```

(下页继续)
## 2. 如果固件的引导加载程序已加密，但通过明文分区表镜像重新烧录了分区表，引导加载程序将无法读取分区表，从而出现以下错误：

<table>
<thead>
<tr>
<th>原因</th>
<th>错误信息</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst:0x3 (SW_RESET), boot:0x13 (SPI_FAST_FLASH_BOOT)</td>
<td>Boot: ESP-IDF v4.0-dev-763-g2c55fae6c-dirty 2nd stage bootloader</td>
</tr>
<tr>
<td>I (56) boot:</td>
<td>Compile time 15:37:14</td>
</tr>
<tr>
<td>I (58) boot:</td>
<td>Enabling RNG early entropy source...</td>
</tr>
<tr>
<td>I (64) boot:</td>
<td>SPI Speed : 40MHz</td>
</tr>
<tr>
<td>I (66) boot:</td>
<td>SPI Flash Size : 4MB</td>
</tr>
<tr>
<td>E (80) flash_parts: partition 0 invalid magic number 0x94f6</td>
<td></td>
</tr>
<tr>
<td>E (86) boot:</td>
<td>Failed to verify partition table</td>
</tr>
<tr>
<td>E (91) boot:</td>
<td>Load partition table error!</td>
</tr>
</tbody>
</table>

## 3. 如果引导加载程序和分区表已加密，但使用明文应用程序镜像重新烧录了应用程序，引导加载程序将无法加载应用程序，从而出现以下错误：

<table>
<thead>
<tr>
<th>原因</th>
<th>错误信息</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst:0x3 (SW_RESET), boot:0x13 (SPI_FAST_FLASH_BOOT)</td>
<td>Boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 2nd stage bootloader</td>
</tr>
<tr>
<td>I (56) boot:</td>
<td>Compile time 15:37:14</td>
</tr>
<tr>
<td>I (58) boot:</td>
<td>Enabling RNG early entropy source...</td>
</tr>
<tr>
<td>I (64) boot:</td>
<td>SPI Speed : 40MHz</td>
</tr>
<tr>
<td>I (68) boot:</td>
<td>SPI Mode : DIO</td>
</tr>
<tr>
<td>I (72) boot:</td>
<td>SPI Flash Size : 4MB</td>
</tr>
<tr>
<td>I (74) boot:</td>
<td>Partition Table:</td>
</tr>
<tr>
<td>I (79) boot:</td>
<td># Label Usage Type ST Offset Length</td>
</tr>
</tbody>
</table>

(下页继续)
4.14.6 ESP32 flash 加密状态

1. 确保您的 ESP32 设备有相关 eFuses 中所示的 flash 加密 eFuse 的默认设置。

要检查您的 ESP32 设备上是否启用了 flash 加密，请执行以下操作之一:

- 将应用示例 security/flash_encryption 暂时重编译到您的设备上。此应用程序会打印 FLASH_CRYPT_CNT eFuse 值，以及是否启用了 flash 加密。
- 查询设备连接的串口名称，在以下命令中将 PORT 替换为串口号名称后运行:

```
espefuse.py -p PORT summary
```

4.14.7 在加密的 flash 中读写数据

ESP32 应用程序代码可以通过调用函数 `esp_flash_encryption_enabled()` 来检查当前是否启用了 flash 加密。此外，设备可以通过调用函数 `esp_get_flash_encryption_mode()` 来识别 flash 加密模式。

一旦启用 flash 加密，使用代码访问 flash 内容时要更加小心。

Flash 加密范围

当 FLASH_CRYPT_CNT eFuse 设置为奇数位的值，所有通过 MMU 的 flash 缓存访问的 flash 内容都将被透明加密，包括:

- Flash 中可执行的应用程序代码 (IROM)。
- 所有存储于 flash 中的只读数据 (DROM)。
- 通过函数 `spi_flash_mmap()` 访问的任意数据。
- ROM 引导加载程序读取的固件引导加载程序镜像。

**重要：** MMU flash 缓存将无条件解密所有数据。Flash 中未加密存储的数据通过 flash 缓存 “被透明解密”，并在软件中存储为随机垃圾数据。

读取加密的 flash

如果需要在不使用 flash 缓存 MMU 映射的情况下读取数据，推荐使用分区读取函数 `esp_partition_read()`，该函数只会解密从加密分区读取的数据。从未加密分区读取的数据不会被解密。这样，软件便能以相同的方式访问加密和未加密的 flash。

也可以使用以下 SPI flash API 函数:

- 通过函数 `esp_flash_read()` 读取不会被解密的原 (加密) 数据。
- 通过函数 `esp_flash_read_encrypted()` 读取和解密数据。

使用非易失性存储器 (NVS) API 存储的数据始终从 flash 加密的角度进行存储和读取解密。如有需要，则由库提供加密功能。详情可参考 NVS 加密。
写入加密的 flash

推荐使用分区写入函数 `esp_partition_write()`。此函数只会在将数据写入加密分区时加密数据，而写入未加密分区的数据不会被加密。通过这种方式，软件可以以相同的方式访问加密和非加密 flash。

也可以使用函数 `esp_flash_write_encrypted()` 加密和写入数据。

此外，esp-idf 应用程序中存在但不支持以下 ROM 函数：

- `esp_rom_spi_flash_write_encrypted` 预加密并将数据写入 flash
- `SPIWrite` 将未加密的数据写入 flash

由于数据是按块加密的，加密数据最小的写入大小为 16 字节，对齐也是 16 字节。

### 4.14.8 更新加密的 flash

**OTA 更新**

如果使用函数 `esp_partition_write()`，对加密分区的 OTA 更新将自动以加密形式写入。

在为已加密设备的 OTA 更新构建应用程序镜像之前，启用项目配置菜单中的 **启动时使能 flash 加密** 选项。请参考 [OTA](#) 获取更多关于 ESP-IDF OTA 更新的信息。

**通过串口更新加密 flash**

通过串行引导加载程序烧录加密设备，需要串行引导加载程序下载接口没有通过 eFuse 被永久禁用。

在开发模式下，推荐的方法是重新烧录更新后的分区。

在发布模式下，如果主机上有存储在 eFuse 中的相同密钥的副本，那么就可以在主机上对文件进行加密，然后进行烧录，具体请参考手动加密文件。

### 4.14.9 关闭 flash 加密

如果意外启用了 flash 加密，则明文数据的 flash 会使 ESP32 无法正常启动。设备将不断重启，并报警 flash read err, 1000 或 invalid header: 0xxxxxxx。

对于开发模式下的 flash 加密，可以通过烧录 FLASH_CRYPT_CNT efuse 来关闭加密。每个芯片仅有 3 次机会，请执行以下步骤：

1. 在项目配置菜单中，禁用 [启动时使能 flash 加密](#) 选项，然后保存并退出。
2. 再次打开项目配置菜单，再次检查你是否已经禁用了该选项，如果这个选项仍被启用，引导加载程序在启动时将立即重新启用加密功能。
3. 在禁用 flash 加密后，通过运行 idf.py flash 来构建和烧录新的引导加载程序和应用程序。
4. 使用 `espefuse.py`（在 `components/esptool_py/esptool` 中）以关闭 FLASH_CRYPT_CNT，运行：

   ```
   espefuse.py burn_efuse FLASH_CRYPT_CNT
   ```

重置 ESP32，flash 加密应处于关闭状态，引导加载程序将正常启动。

### 4.14.10 Flash 加密的要点

- 使用 AES-256 加密 flash。Flash 加密密钥存储于芯片内部的 `flash_encryption` eFuse 中，并（默认）受保护，防止软件访问。
• Flash 加密算法采用的是 AES-256，其中密钥随着 flash 的每个 32 字节块的偏移地址“调整”。这意味着，每个 32 字节块（2 个连续的 16 字节 AES 块）使用从 flash 加密密钥中产生的一个特殊密钥进行加密。

• 通过 ESP32 的 flash 缓存映射功能，flash 可支持透明访问——任何映射到地址空间的 flash 区域在读取时都被透明地解密。

为便于访问，某些数据分区最好保持未加密状态，或者也可使用对已加密数据无效的 flash 友好型更新算法。由于 NVS 库无法与 flash 加密直接兼容，因此无法加密非易失性存储器的 NVS 分区。详情可参见 NVS 加密。

• 如果以后可能需要启用 flash 加密，则编程人员在编写使用加密 flash 代码时需小心谨慎。

• 如果已启用安全启动，重新烧录加密设备的文字加载程序则需要“可重新烧录”的安全启动摘要（可参考 flash 加密与安全启动）。

启用 flash 加密将增大引导加载程序，因此可能需更新分区表偏移量。请参考引导加载程序大小。

重要：在首次启动加密过程中，请勿切断 ESP32 的电源。如果电源被切断，flash 的内容将被破坏，并需要重新烧录未加密数据。而这类重新烧录将不超过烧录限制次数。

4.14.11 Flash 加密的局限性

flash 加密可以保护固件，防止未经授权的读取与修改。了解 flash 加密系统的局限之处亦十分重要：

• Flash 加密功能与密钥同样稳固。因而，推荐您首次启动设备时在设备上生成密钥（默认行为）。如果在设备外生成密钥，请确保遵循正确的后续步骤，不要在所有生产设备之间使用相同的密钥。

• 并非所有数据都是加密存储。因而在 flash 上存储数据时，请检查您使用的存储方式（库、API 等）是否支持 flash 加密。

• Flash 加密无法防止攻击者获取 flash 的高层次布局信息。这是因为每对相邻的 16 字节 AES 块都使用相邻的 AES 密钥。当这些相邻的 16 字节块中包含相同内容时（如空白或填充区域），这些字节块将加密以产生匹配的加密块对。这让攻击者可在加密设备间进行高层次对比（例如，确认两设备是否可能是运行相同的固件版本）。

• 出于相同原因，攻击者始终可获取一对相邻的 16 字节块（32 字节对齐）时，包含相同的 16 字节序列。因此，在 flash 上存储敏感数据时应牢记这点，可进行相关设置避免该情况发生（可使用计数器字节或每 16 字节设置不同的值即可）。具体请参考 NVS 加密。

• 单独使用 flash 加密可能无法防止攻击者修改本设备的固件。为防止设备上运行未经授权的固件，可搭配 flash 加密使用安全启动。

4.14.12 Flash 加密与安全启动

推荐 flash 加密与安全启动搭配使用。但是，如果已启用安全启动，则重新烧录设备时会受到其他限制：

• 如果新的应用程序已使用安全启动签名密钥正确签名，则 OTA 更新 不受限制。

• 只有当选择可再次烧录 安全启动模式，且安全启动密钥已预生成并烧录至 ESP32 时（可参见 安全启动），明文串行 flash 更新 才可能实现。在该配置下，idf.py bootloader 将生成简化的引导加载程序和安全启动摘要文件，以偏移量 0x0 处进行烧录。当进行明文串行重新烧录步骤时，需在烧录其他明文数据前重新烧录此文件。

• 如果未重新烧录引导加载程序，则仍然可以使用预生成的 flash 加密密钥重新烧录。重新烧录引导加载程序时，需在安全启动配置中启用相同的可重新烧录 选项。

4.14.13 Flash 加密的高级功能

以下部分介绍了 flash 加密的高级功能。
加密分区标志

部分分区默认为已加密。通过在分区的标志字段中添加“encrypted”标志，可在分区表描述中将其他分区标记为需要加密。在这些标记分区中的数据会和应用程序分区一样视为加密数据。

<table>
<thead>
<tr>
<th>Name, Type, SubType, Offset, Size, Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvs, data, nvs, 0x9000, 0x6000</td>
</tr>
<tr>
<td>phy_init, data, phy, 0xf000, 0x1000</td>
</tr>
<tr>
<td>factory, app, factory, 0x10000, 1M</td>
</tr>
<tr>
<td>secret_data, 0x40, 0x01, 0x20000, 256K, encrypted</td>
</tr>
</tbody>
</table>

请参考分区分表获取更多关于分区表描述的具体信息。

关于分区加密您还需要了解以下信息:

- 默认分区表都不包含任何加密数据分区。
- 启用 flash 加密后，“app”分区一般都视为加密分区，因此无需标记。
- 如果未启用 flash 加密，则“encrypted”标记无效。
- 将可选 phy 分区标记为“encrypted”，可以防止物理访问读取或修改 phy_init 数据。
- nvs 分区无法标记为“encrypted”因为NVS库与flash加密不直接兼容。

启用 UART 引导加载程序加密/解密

在第一次启动时，flash 加密过程默认会烧录以下 eFuse:

- DISABLE_DL_ENCRYPT 在 UART 引导加载程序启动模式下运行时，禁止 flash 加密操作。
- DISABLE_DL_DECRYPT 在 UART 引导加载程序模式下运行时，禁止透明 flash 解密 (即使 eFuse FLASH_CRYPT_CNT 已设置为在正常操作中启用 flash 解密)。
- DISABLE_DL_CACHE 在 UART 引导加载程序模式下运行时禁止整个 MMU flash 缓存。

为了能启用这些功能，可在首次启动前仅烧录部分 eFuse，并用未设置值 0 写保护其他部分。例如：

```bash
espefuse.py --port PORT burn_efuse DISABLE_DL_DECRYPT
espefuse.py --port PORT write_protect_efuse DISABLE_DL_ENCRYPT
```

重要：保持DISABLE_DL_DECRYPT 未设置 (0) 会使 flash 加密无效。

对芯片具有物理访问权限的攻击者会使用 UART 引导加载程序模式（使用自定义存根代码）读取 flash 的内容。

设置 FLASH_CRYPT_CONFIG

FLASH_CRYPT_CONFIG eFuse 决定 flash 加密密钥中密钥偏移“调整”的位数。详情可参考Flash 加密算法。

首次启动固件引导加载程序时，该值始终设置为最大值 0xF。

可手动写入这些 eFuse，并在首次启动前对其进行写保护，以便选择不同的调整值。但不推荐该操作。

强烈建议在FLASH_CRYPT_CONFIG 未设置时，不要对其进行写保护。否则，它的值将永久为零，而 flash 加密密钥中则无调整位。这导致 flash 加密算法等同于 AES ECB 模式。

JTAG 调试

默认情况下，当启用 flash 加密（开发或发布模式）时，将通过 eFuse 禁用 JTAG 调试。引导加载程序在首次启动时执行此操作，同时启用 flash 加密。

请参考JTAG 与闪存加密和安全引导 了解更多关于使用 JTAG 调试与 flash 加密的信息。
### 手动加密文件

手动加密或解密文件需要在 eFuse 中预烧录 flash 加密密钥（请参阅使用主机生成的密钥）并在主机上保留一份副本。如果 flash 加密配置在开发模式下，那么则不需要保留密钥的副本或遵循这些步骤，可以使用更简单的重新烧录更新后的分区步骤。

密钥文件应该是单个原始二进制文件（例如: key.bin）。

例如，以下是将文件 build/my-app.bin 进行加密，烧录到偏移量 0x10000 的步骤，运行 espsecure.py，如下所示:

```bash
espsecure.py encrypt_flash_data --keyfile /path/to/key.bin --address 0x10000 -- --output my-app-ciphertext.bin build/my-app.bin
```

然后可以使用 esptool.py 将文件 my-app-ciphertext.bin 写入偏移量 0x10000。关于为 esptool.py 推荐的所有命令行选项，请查看 idf.py build 成功时打印的输出。

### 备注:

如果 ESP32 在启动时无法识别烧录进去的密文文件，请检查密钥是否匹配以及命令行参数是否完全匹配，包括偏移量是否正确。

如果您的 ESP32 在 eFuse 中使用了非默认的 FLASH_CRYPT_CONFIG 值，那么则需要向 espsecure.py 传递 --flash_crypt_conf 参数以设置匹配的值。如果设备自己设置了 flash 加密就不会出现这种情况，但如果手动烧录 eFuse 来启用 flash 加密就可能发生这种情况。

```bash
espsecure.py decrypt_flash_data 命令可以使用同样的选项（和不同的输入/输出文件）来解密 flash 密文或之前加密的文件。
```

### 4.14.14 技术细节

以下章节将提供 flash 加密操作的相关信息。

#### Flash 加密算法

- AES-256 在 16 字节的数据块上运行。Flash 加密引擎在 32 字节的数据（2 个串行 AES 块）上加密或解密数据。
- Flash 加密的主密钥存储于 flash_encryption eFuse 中，默认受保护防止进一步写入或软件读取。
- AES-256 密钥大小为 256 位（32 字节），从 flash_encryption eFuse 中读取。与 flash_encryption 中的存储顺序相比，硬件 AES 引擎使用的是相反的字节顺序的密钥。
  - 如果 CODING_SCHEME eFuse 设置为 0（默认“无”编码方案），则 eFuse 密钥块为 256 位，且密钥按原方式存储（反字节序）。
  - 如果 CODING_SCHEME eFuse 设置为 1 (3/4 编码)，则 eFuse 密钥块为 192 位（反字节序），信息熵总量减少。硬件 flash 加密仍在 256 字节密钥上运行，在读取后（字节序未反向），密钥扩展为 key = key[0:255] + key[64:127]。
- Flash 加密中使用了逆向 AES 算法，因此 flash 加密的“加密”操作相当于 AES 解密，而其“解密”操作则相当于 AES 加密。这是为了优化性能，不会影响算法的有效性。
- 每个 32 字节块（2 个相邻的 16 字节 AES 块）都由一个特殊的密钥进行加密。该密钥由 flash_encryption 中 flash 加密的主密钥产生，并随 flash 中该字节块的偏移进行 XOR 运算（一次“密钥调整”）。
- 具体调整量取决于 FLASH_CRYPT_CONFIG eFuse 的设置。该 eFuse 共 4 位，每位可对特定范围的密钥位进行 XOR 运算：
  - Bit 1，对密钥的 0-66 位进行 XOR 运算。
  - Bit 2，对密钥的 67-131 位进行 XOR 运算。
  - Bit 3，对密钥的 132-194 位进行 XOR 运算。
  - Bit 4，对密钥的 195-256 位进行 XOR 运算。
4.15 Hardware Abstraction

Hardware abstraction in ESP-IDF are a group of API that allow users to control peripherals at differing levels of abstraction, as opposed to interfacing with hardware using only the ESP-IDF drivers. ESP-IDF Hardware abstraction will likely be useful for users writing high performance bare-metal drivers, or for those attempting to port an ESP chip to another platform.

This guide is split into the following sections:

1. Architecture
2. LL (Low Level) Layer
3. HAL (Hardware Abstraction Layer)

警告: Hardware abstraction API (excluding the driver and xxx_types.h) should be considered an experimental feature, thus cannot be considered public API. Hardware abstraction API do not adhere to the API name changing restrictions of ESP-IDF’s versioning scheme. In other words, it is possible that Hardware Abstraction API may change in between non-major release versions.

备注: Although this document mainly focuses on hardware abstraction of peripherals (e.g., UART, SPI, I2C), certain layers of hardware abstraction extend to other aspects of hardware as well (e.g., some of the CPU’s features are partially abstracted).

4.15.1 Architecture

Hardware abstraction in ESP-IDF is comprised of the following layers, ordered from low level (closer to hardware) to high level (further away from hardware) of abstraction.

- Low Level (LL) Layer
- Hardware Abstraction Layer (HAL)
- Driver Layers

The LL Layer, and HAL are entirely contained within the hal component. Each layer is dependent on the layer below it (i.e., driver depends on HAL, HAL depends on LL, LL depends on the register header files).

For a particular peripheral xxx, its hardware abstraction will generally consist of the header files described in the table below. Files that are Target Specific will have a separate implementation for each target (i.e., a separate copy for each chip). However, the #include directive will still be target-independent (i.e., will be the same for different targets) as the build system will automatically include the correct version of the header and source files.
## 4.15.2 LL (Low Level) Layer

The primary purpose of the LL Layer is to abstract away register field access into more easily understandable functions. LL functions essentially translate various in/out arguments into the register fields of a peripheral in the form of get/set functions. All the necessary bit shifting, masking, offsetting, and endianness of the register fields should be handled by the LL functions.

```c
//Inside xxx_ll.h

static inline void xxx_ll_set_baud_rate(xxx_dev_t *hw,
    xxx_ll_clk_src_t clock_source,
    uint32_t baud_rate) {
    uint32_t src_clk_freq = (source_clk == XXX_SCLK_APB) ? APB_CLK_FREQ : REF_CLK_FREQ;
    uint32_t clock_divider = src_clk_freq / baud;
    // Set clock select field
    hw->clk_div_reg.divider = clock_divider >> 4;
    // Set clock divider field
```
The code snippet above illustrates typical LL functions for a peripheral `xxx`. LL functions typically have the following characteristics:

- All LL functions are defined as `static inline` so that there is minimal overhead when calling these functions due to compiler optimization. These functions are not guaranteed to be inlined by the compiler, so any LL function that will be called when the cache is disabled (e.g. from an IRAM ISR context) should be marked with `__attribute__((always_inline)).`
- The first argument should be a pointer to a `xxx_dev_t` type. The `xxx_dev_t` type is a structure representing the peripheral’s registers, thus the first argument is always a pointer to the starting address of the peripheral’s registers. Note that in some cases where the peripheral has multiple channels with identical register layouts, `xxx_dev_t *hw` may point to the registers of a particular channel instead.
- LL functions should be short and in most cases are deterministic. In other words, the worst case runtime of the LL function can be determined at compile time. Thus, any loops in LL functions should be finite bounded; however, there are currently a few exceptions to this rule.
- LL functions are not thread safe, it is the responsibility of the upper layers (driver layer) to ensure that registers or register fields are not accessed concurrently.

### 4.15.3 HAL (Hardware Abstraction Layer)

The HAL layer models the operational process of a peripheral as a set of general steps, where each step has an associated function. For each step, the details of a peripheral’s register implementation (i.e., which registers need to be set/read) are hidden (abstracted away) by the HAL. By modeling peripheral operation as a set of functional steps, any minor hardware implementation differences of the peripheral between different targets or chip versions can be abstracted away by the HAL (i.e., handled transparently). In other words, the HAL API for a particular peripheral will remain mostly the same across multiple targets/chip versions.

The following HAL function examples are selected from the Watchdog Timer HAL as each function maps to one of the steps in a WDT’s operation life cycle, thus illustrating how a HAL abstracts a peripheral’s operation into functional steps.

```c
// Initialize one of the WDTs
void wdt_hal_init(wdt_hal_context_t *hal, wdt_inst_t wdt_inst, uint32_t prescaler, __bool enable_intr);

// Configure a particular timeout stage of the WDT
void wdt_hal_config_stage(wdt_hal_context_t *hal, wdt_stage_t stage, uint32_t timeout, wdt_stage_action_t behavior);

// Start the WDT
void wdt_hal_enable(wdt_hal_context_t *hal);

// Feed (i.e., reset) the WDT
void wdt_hal_feed(wdt_hal_context_t *hal);

// Handle a WDT timeout
void wdt_hal_handle_intr(wdt_hal_context_t *hal);

// Stop the WDT
void wdt_hal_disable(wdt_hal_context_t *hal);

// De-initialize the WDT
void wdt_hal_deinit(wdt_hal_context_t *hal);
```
HAL functions will generally have the following characteristics:

- The first argument to a HAL function has the `xxx_hal_context_t *` type. The HAL context type is used to store information about a particular instance of the peripheral (i.e., the context instance). A HAL context is initialized by the `xxx_hal_init()` function and can store information such as the following:
  - The channel number of this instance
  - Pointer to the peripheral’s (or channel’s) registers (i.e., a `xxx_dev_t *` type)
  - Information about an ongoing transaction (e.g., pointer to DMA descriptor list in use)
  - Some configuration values for the instance (e.g., channel configurations)
  - Variables to maintain state information regarding the instance (e.g., a flag to indicate if the instance is waiting for transaction to complete)

- HAL functions should not contain any OS primitives such as queues, semaphores, mutexes, etc. All synchronization/concurrency should be handled at higher layers (e.g., the driver).
- Some peripherals may have steps that cannot be further abstracted by the HAL, thus will end up being a direct wrapper (or macro) for an LL function.
- Some HAL functions may be placed in IRAM thus may carry an `IRAM_ATTR` or be placed in a separate `xxx_hal_iram.c` source file.

### 4.16 High-Level Interrupts

The Xtensa architecture has support for 32 interrupts, divided over 7 levels (levels 1 to 7, with 7 being an NMI), plus an assortment of exceptions. On the ESP32, the interrupt mux allows most interrupt sources to be routed to these interrupts using the interrupt allocator. Normally, interrupts will be written in C, but ESP-IDF allows high-level interrupts to be written in assembly as well, resulting in very low interrupt latencies.

#### 4.16.1 Interrupt Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Symbol</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N/A</td>
<td>Exception and level 0 interrupts. Handled by ESP-IDF</td>
</tr>
<tr>
<td>2-3</td>
<td>N/A</td>
<td>Medium level interrupts. Handled by ESP-IDF</td>
</tr>
<tr>
<td>4</td>
<td><code>xt_highint4</code></td>
<td>Free to use (See 1)</td>
</tr>
<tr>
<td>5</td>
<td><code>xt_highint5</code></td>
<td>Normally used by ESP-IDF debug logic (See 1)</td>
</tr>
<tr>
<td>NMI</td>
<td><code>xt_nmi</code></td>
<td>Free to use</td>
</tr>
<tr>
<td>dbg</td>
<td><code>xt_debugexception</code></td>
<td>Debug exception. Called on e.g. a BREAK instruction. (See 2)</td>
</tr>
</tbody>
</table>

The following notes give more information about the items in the tables above.

1. ESP-IDF debug logic can be configured to run on `xt_highint4` or `xt_highint5` in `CONFIG_ESP_SYSTEM_CHECK_INT_LEVEL`. Bluetooth’s interrupt can be configured to run on level 4 by enabling `CONFIG_BTDM_CTRL_HLI`. If `CONFIG_BTDM_CTRL_HLI` is enabled, ESP-IDF debug logic must be running on level 5 interrupt.

2. If `CONFIG_BTDM_CTRL_HLI` is enabled, `xt_debugexception` is used to fix live lock issue in ESP32 ECO3.

Using these symbols is done by creating an assembly file (suffix `.S`) and defining the named symbols, like this:

```assembly
.section .iram1,"ax"
.global xt_highint5
.type xt_highint5, @function
.align 4
xt_highint5:
    ... your code here
rser a0, EXCSAVE_5
rfi 5
```

For a real-life example, see the `esp_system/port/soc/esp32/highint_hdl.S` file; the panic handler interrupt is implemented there.
4.16.2 Notes

- Do not call C code from a high-level interrupt; as these interrupts are run from a critical section, this can cause the target to crash. Note that although the panic handler interrupt does call normal C code, this exception is allowed due to the fact that this handler never returns (i.e., the application will not continue to run after the panic handler). So breaking C code execution flow is not a problem.

  When `CONFIG_BTDM_CTRL_HLI` is enabled, C code is also called from a high-level interrupt, this is possible thanks to some additional protection added to it.

- Make sure your assembly code gets linked in. Indeed, as the free-to-use symbols are declared as weak, the linker may discard the file containing the symbol. This will happen if the only symbol defined, or used, from the user file is the `xt_*` free-to-use symbol. To avoid this, in the assembly file containing the `xt_*` symbol, define another symbol, like:

  ```
  .global ld_include_my_isr_file
  ld_include_my_isr_file:
  ```

  Here it is called `ld_include_my_isr_file` but can have any name, as long as it is not defined anywhere else in the project.

  Then, in the component `CMakeLists.txt`, add this name as an unresolved symbol to the ld command line arguments:

  ```
  target_link_libraries(${COMPONENT_TARGET} "-u ld_include_my_isr_file")
  ```

  This should cause the linker to always include the file defining `ld_include_my_isr_file`, causing the ISR to always be linked in.

- High-level interrupts can be routed and handled using `esp_intr_alloc()` and associated functions. The handler and handler arguments to `esp_intr_alloc()` must be NULL, however.

- In theory, medium priority interrupts could also be handled in this way. ESP-IDF does not support this yet.

- To check Xtensa instruction set architecture (ISA), please refer to Xtensa ISA Summary.

4.17 JTAG 调试

本文将介绍如何安装 ESP32 的 OpenOCD 调试环境，以及如何使用 GDB 来调试 ESP32 的应用程序。本文结构如下:

引言 介绍本指南主旨。
工作原理 介绍 ESP32、JTAG（Joint Test Action Group）接口、OpenOCD 和 GDB 如何相互连接，从而实现 ESP32 的调试功能。
选择 JTAG 适配器 介绍有关 JTAG 硬件适配器的选择及参照标准。
安装 OpenOCD 介绍如何安装官方预编译好的 OpenOCD 软件包并验证是否安装成功。
配置 ESP32 目标板 介绍如何设置 OpenOCD 软件并安装 JTAG 硬件，两项共同构成调试目标。
启动调试器 介绍如何从 Eclipse 集成开发环境 启动 GDB 调试时会话。
调试范例 如果您不熟悉 GDB，请查看此小节以获取 Eclipse 集成开发环境 提供的调试示例。
从源码构建 OpenOCD 介绍如何在 Windows、Linux 和 macOS 操作系统上从源码构建 OpenOCD。
注意事项和补充内容 介绍使用 OpenOCD 和 GDB 通过 JTAG 接口调试 ESP32 时的注意事项和补充内容。

4.17.1 引言

ESP32 具有两个强大的 Xtensa 内核，支持多种程序架构。ESP-IDF 自带的 FreeRTOS 操作系统支持多核抢占式调度，允许用户以更加直观的方式编写软件。与此相对地，由于缺乏合适的工具，简便的编程方式也会给程序的调试带来困难。
乐鑫已完成 OpenOCD 移植，以支持 ESP32 处理器和多核 FreeRTOS 架构（大多数 ESP32 应用程序的基础）。此外，乐鑫还提供了一些 OpenOCD 本身并不支持的工具，以进一步丰富调试功能。

本文将介绍如何在 Linux、Windows 和 macOS 环境下为 ESP32 安装 OpenOCD，并使用 GDB 进行软件调试。除部分安装流程有所不同外，所有操作系统的软件用户界面和使用流程都是相同的。

备注：本文使用的图片素材来自于 Ubuntu 16.04 LTS 上 Eclipse Neon 3 软件的截图，不同的操作系统（Windows、macOS 或 Linux）或不同的 Eclipse 软件版本在用户界面上可能会有细微差别。

4.17.2 工作原理

通过 JTAG（Joint Test Action Group）接口使用 OpenOCD 调试 ESP32 时所需要的关键软件和硬件包括 xtensa-esp32elf-gdb 调试器、OpenOCD 片上调试器和连接到 ESP32 目标的 JTAG 适配器，如下图“Application Loading and Monitoring”标志所示。

“Application Loading and Monitoring”标志显示一组关键的软件和硬件组件，可用于编译、构建和烧写应用程序到 ESP32 上，以及监视来自 ESP32 的运行诊断信息。

Eclipse 环境集成了 JTAG 调试和应用程序加载、监视的功能，使得软件从编写、编译、加载到调试的迭代过程变得更加快速简单。Eclipse IDE 及其集成的调试软件均适用于 Windows、Linux 和 macOS 平台。根据用户喜好，除了使用 Eclipse 集成开发环境，还可以直接在命令行终端运行 debugger 和 idf.py build。

若使用 ESP-WROOM-KIT 开发板，由于其板载 FT232H 芯片，仅需一根 USB 线即可连接 PC 与 ESP32。FT232H 提供了两路 USB 通道，一路连接到 JTAG，另一路连接到 UART。

4.17.3 选择 JTAG 适配器

上手 JTAG 最快速便捷的方式是使用 ESP-WROOM-KIT 开发板，因为它板载了 JTAG 调试接口，无需使用外部 JTAG 硬件适配器和额外线缆来连接 JTAG 与 ESP32。ESP-WROOM-KIT 采用 FT2232H 提供的 JTAG 接口，可以稳定运行在 20 MHz 的时钟频率，外接的适配器很难达到这个速度。

如果您想使用单独的 JTAG 适配器，请确保其与 ESP32 的电平电压和 OpenOCD 软件都兼容。ESP32 使用的是业界标准的 JTAG 接口，它未使用（实际上也并不需要）TRST 信号脚。JTAG 使用的 IO 管脚由
Chapter 4. API 指南

VDD_3P_RTC 电源管脚供电（通常连接到外部 3.3 V 的电源轨），因此 JTAG 硬件适配器的管脚需要能够在该电压范围内正常工作。

在软件方面，OpenOCD 支持相当多数量的 JTAG 适配器，请参阅 OpenOCD 支持的适配器列表（请注意这一列表并不完整），其中还列出了兼容 SWD 接口的适配器，但请注意，ESP32 目前并不支持 SWD。此外，硬编码为只支持特定产品线的 JTAG 适配器也无法在 ESP32 上工作，例如仅针对 STM32 系列产品的 ST-LINK 适配器。

保证 JTAG 正常工作需要连接的信号线包括：TDI、TDO、TCK、TMS 和 GND。一些 JTAG 适配器还需要 ESP32 提供一路电源到适配器的某个管脚上（比如 Vtar），用于设置适配器的工作电压。您也可以选择将 SRST 信号线连接到适配器的 CH_PD 管脚上，但请注意，目前 OpenOCD 对该信号线提供的支持相当有限。

ESP-Prog 中展示了使用外部电路板进行调试的实例，方法是将其连接到 ESP32 的 JTAG 管脚上。

4.17.4 安装 OpenOCD

如果您已经按照 快速入门 完成了 ESP-IDF 及其 CMake 构建系统的安装，那么 OpenOCD 已经被默认安装到了您的开发系统中。在 设置开发环境 结束后，您应该能够在终端中运行如下 OpenOCD 命令：

```
openocd --version
```

终端会输出以下信息（实际版本号可能会更新）：

```markdown
Open On-Chip Debugger v0.10.0-esp32-20190708 (2019-07-08-11:04)
Licensed under GNU GPL v2
For bug reports, read
https://openocd.org/doc/doxygen/bugs.html
```

您还可以检查 OPENOCD_SCRIPTS 环境变量的值，以确认 OpenOCD 配置文件的路径。Linux 和 macOS 用户可以在终端输入 `echo $OPENOCD_SCRIPTS`，Windows 用户需要输入 `echo %OPENOCD_SCRIPTS%`。如果终端输出了有效路径，则表明您已经正确安装 OpenOCD。

如果无法执行上述步骤，请再次阅读快速入门手册，参考 设置安装工具 章节。

备注：另外也可以从源代码编译 OpenOCD 工具，详细信息请参阅 源码构建 OpenOCD 章节。

4.17.5 配置 ESP32 目标板

OpenOCD 安装完成后就可以配置 ESP32 目标（即带 JTAG 接口的 ESP32 板），具体分为以下三个步骤：

- 配置并连接 JTAG 接口
- 运行 OpenOCD
- 上传待调试的应用程序

配置并连接 JTAG 接口

此步骤取决于使用的 JTAG 和 ESP32 板，参考以下两种情况。

配置 ESP-WROVER-KIT 上的 JTAG 接口

所有版本的 ESP-WROVER-KIT 板子都内置了 JTAG 调试功能，要使其正常工作，还需要设置相关跳线来启用 JTAG 功能，设置 SPI 闪存电压和配置 USB 驱动程序。具体步骤请参考以下说明。
配置硬件

- 根据ESP-WROVER-KIT V4.1入门指南文档中设置选项章节所描述的信息，设置JP8便可以启用JTAG功能。
- 检查ESP32上用于JTAG通信的引脚是否被接到了其它硬件上，这可能会影响JTAG的工作。

<table>
<thead>
<tr>
<th>ESP32管脚</th>
<th>JTAG信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTDO/GPIO15</td>
<td>TDO</td>
</tr>
<tr>
<td>MTDI/GPIO12</td>
<td>TDI</td>
</tr>
<tr>
<td>MTCK/GPIO13</td>
<td>TCK</td>
</tr>
<tr>
<td>MTMS/GPIO14</td>
<td>TMS</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>

配置USB驱动 安装和配置USB驱动，这样OpenOCD才能够与ESP-WROVER-KIT板上的JTAG接口通信，并且使用UART接口上传待烧写的镜像文件。请根据你的操作系统按照以下步骤进行安装配置。

备注：ESP-WROVER-KIT使用了FT2232芯片实现了JTAG适配器，所以以下说明同样适用于其他基于FT2232的JTAG适配器。

Windows

1. 使用标准USB A / micro USB B线将ESP-WROVER-KIT与计算机相连接，并打开板子的电源。
2. 等待Windows识别出ESP-WROVER-KIT并且为其安装驱动。如果驱动没有被自动安装，请前往官网下载并手动安装。
3. 从Zadig官网下载Zadig工具（Zadig_X.X.exe）并运行。
4. 在Zadig工具中，进入“Options”菜单中选中“List All Devices”。
5. 检查设备列表，其中应该包含两条与ESP-WROVER-KIT相关的条目：“Dual RS232- HS (Interface 0)”和“Dual RS232- HS (Interface 1)”。驱动的名字应该是“FTDIBUS (vxxxx)”并且USB ID为：04036010。

图36: 在Zadig工具中配置JTAG USB驱动

6. 第一个设备“Dual RS232-HS (Interface 0)”连接到了ESP32的JTAG端口，此设备原来的“FTDIBUS (vxxxx)”驱动需要替换成“WinUSB (v6xxxxx)”。为此，请选择“Dual RS232-HS (Interface 0)”并将驱动重新安装为“WinUSB (v6xxxxx)”，具体可以参考上图。
Linux

1. 使用标准 USB A / micro USB B 线将 ESP-WROVER-KIT 与计算机相连接，并打开板子的电源。

2. 打开终端，输入 `ls -l /dev/ttyUSB*` 命令检查操作系统是否能够识别板子的 USB 端口。类似识别结果如下：

   ```
   user-name@computer-name:~/esp$ ls -l /dev/ttyUSB*
   crw-rw---- 1 root dialout 188, 0 Jul 10 19:04 /dev/ttyUSB0
   crw-rw---- 1 root dialout 188, 1 Jul 10 19:04 /dev/ttyUSB1
   ```

3. 根据 OpenOCD README 文档中“Permissions delegation”小节的介绍，设置这两个 USB 端口的访问权限。

4. 注销并重新登录 Linux 系统，然后重新插入板子的电源使之前的改动生效。在终端再次输入 `ls -l /dev/ttyUSB*` 命令进行验证，查看这两个设备的组所有是否已经从 dialout 更改为 plugdev：

   ```
   user-name@computer-name:~/esp$ ls -l /dev/ttyUSB*
   crw-rw---- 1 root plugdev 188, 0 Jul 10 19:07 /dev/ttyUSB0
   crw-rw---- 1 root plugdev 188, 1 Jul 10 19:07 /dev/ttyUSB1
   ```

如果看到类似的输出结果，并且你也是 plugdev 组的成员，那么设置工作就完成了。具有较低编号的 `/dev/ttyUSBn` 接口用于 JTAG 通信，另一路接口被连接到 ESP32 的串口 (UART)，用于上传应用程序映像给 ESP32 进行烧写。

现在，ESP-WROVER-KIT 的 JTAG 接口应该可以被 OpenOCD 使用了，想要进一步设置调试环境，请前往运行 OpenOCD 章节。

MacOS 在 macOS 上，同时使用 FT2232 的 JTAG 接口和串口还需另外进行其它操作。当操作系统加载 FTDI 串口驱动的时候，它会对 FT2232 芯片的两个通道做相同的操作。但是，这两个通道中只有一个是被用作串口，而另一个用于 JTAG，如果操作系统已经为用于 JTAG 的通道加载了 FTDI 串口驱动的话，OpenOCD 将无法连接到芯片。有两个方法可以解决这个问题：

1. 在启动 OpenOCD 之前手动卸载 FTDI 串口驱动程序，然后启动 OpenOCD，再加载串口驱动程序。
2. 修复 FTDI 驱动程序的配置，使其不会为 FT2232 芯片的通道 A 进行自我加载，该通道用于 ESP-WROVER-KIT 板上的 JTAG 通道。

手动卸载驱动程序

1. 从 FTDI 官网 安装驱动。
2. 使用 USB 线连接 ESP-WROVER-KIT。
3. 卸载串口驱动

   ```
   sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
   ```

有时，您可能还需要卸载苹果的 FTDI 驱动：

- macOS < 10.15:

   ```
   sudo kextunload -b com.apple.driver.AppleUSBFTDI
   ```

- macOS 10.15:

   ```
   sudo kextunload -b com.apple.DriverKit-AppleUSBFTDI
   ```
警告：对于 FTDI 驱动，如果使用串口的通道不正确，则可能会导致内核崩溃。ESP-WROVER-KIT 将通道 A 用于 JTAG，通道 B 用于串口。

4. 运行 OpenOCD:

```bash
.. include:: esp32.inc
:start-after: run-openocd
:end-before: ---
```

5. 在另一个终端窗口，再一次加载 FTDI 串口驱动:

```bash
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

备注：如果你需要重启 OpenOCD，则无需再次卸载 FTDI 驱动程序，只需停止 OpenOCD 并再次启动它。只有在重新连接 ESP-WROVER-KIT 或者切换了电源的情况下才需要再次卸载驱动。

你也可以根据自身需求，将此过程包装进 shell 脚本中。

修改 FTDI 驱动 简而言之，这种方法需要修改 FTDI 驱动程序的配置文件，这样可以防止为 FT2232H 的通道 B 自动加载串口驱动。

备注：其他板子可能将通道 A 用于 JTAG，因此请谨慎使用此选项。

警告：此方法还需要操作系统禁止对驱动进行签名验证，因此可能无法被所有的用户所接受。

1. 使用文本编辑器打开 FTDI 驱动的配置文件 (注意 sudo):

   ```bash
   sudo nano /Library/Extensions/FTDIUSBSerialDriver.kext/Contents/Info.plist
   ```

2. 找到并删除以下几行:

   ```xml
   <key>FT2232H_B</key>
   <dict>
   <key>CFBundleIdentifier</key>
   <string>com.FTDI.driver.FTDIUSBSerialDriver</string>
   <key>IOClass</key>
   <string>FTDIUSBSerialDriver</string>
   <key>IOProviderClass</key>
   <string>IOUSBInterface</string>
   <key>bConfigurationValue</key>
   <integer>1</integer>
   <key>bInterfaceNumber</key>
   <integer>1</integer>
   <key>bcdDevice</key>
   <integer>1792</integer>
   <key>idProduct</key>
   <integer>24592</integer>
   <key>idVendor</key>
   <integer>1027</integer>
   </dict>
   ```

3. 保存并关闭文件

4. 禁用驱动的签名认证:
   1. 点击苹果的 logo，选择 “Restart…”
   2. 重启后当听到响铃时，立即按下键盘上的 CMD+R 组件键
   3. 进入恢复模式后，打开终端
   4. 运行命令:
第4章. API

```bash
csrutil enable --without kext
```

5. 再一次重启系统
完成这些步骤后，可以同时使用串口和 JTAG 接口了。
想要进一步设置调试环境，请前往运行 OpenOCD 章节。

### 配置其他 JTAG 接口
关于适配 OpenOCD 和 ESP32 的 JTAG 接口选择问题，请参考选择 JTAG 适配器 章节，确保 JTAG 适配器能够与 OpenOCD 和 ESP32 一同工作。然后按照以下三个步骤进行设置，使其正常工作。

### 配置硬件
1. 找到 JTAG 接口和 ESP32 板上需要相互连接并建立通信的所有管脚或信号。

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th>JTAG 信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTDO / GPIO15</td>
<td>TDO</td>
</tr>
<tr>
<td>MTDI / GPIO12</td>
<td>TDI</td>
</tr>
<tr>
<td>MTMS / GPIO13</td>
<td>TCK</td>
</tr>
<tr>
<td>MTMS / GPIO14</td>
<td>TMS</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>

2. 检查 ESP32 上用于 JTAG 通信的管脚是否被连接到了其它硬件上，这可能会影响 JTAG 的工作。
3. 连接 ESP32 和 JTAG 接口上的管脚或信号。

### 配置驱动
您可能还需要安装软件驱动，才能使 JTAG 在计算机上正常工作，请参阅您所使用的 JTAG 适配器的有关文档，获取相关详细信息。

### 连接
将 JTAG 接口连接到计算机，打开 ESP32 和 JTAG 接口板上的电源，然后检查计算机是否可以识别到 JTAG 接口。
如需继续设置调试环境，请前往运行 OpenOCD 章节。

### 运行 OpenOCD
配置完目标并将其连接到电脑后，即可启动 OpenOCD。
打开终端，按照快速入门指南中的设置好开发环境 章节进行操作，然后运行如下命令，以启动 OpenOCD（该命令适用于 Windows、Linux 和 macOS）:

```bash
openocd -f board/esp32-wrover-kit-3.3v.cfg
```

### 备注：
上述命令中 -f 选项后跟的配置文件专用于板载 ESP32-WROOM-32 模组的 ESP-WROVER-KIT 开发板。基于具体使用的硬件，您可能需要选择不同的配置文件，具体内容请参阅根据目标芯片配置 OpenOCD。

例如，对于带有用于 JTAG 连接的 FT2232H 或 FT232H 芯片的定制板，或带有 ESP-Prog 的定制板，可使用 board/esp32c3-ftdi.cfg。

现在您应该可以看到如下输出（此日志来自板载 ESP32-WROOM-32 模组的 ESP-WROVER-KIT 开发板）:
• 如果出现提示权限问题的错误，请打开 ~/esp/openocd-esp32 目录，参阅 OpenOCD README 文件中关于“Permissions delegation”的说明。
• 如果遇到无法找到配置文件的错误，例如 Can't find board/esp32-wrover-kit-3.3v. cfg，请检查 OPENOCD_SCRIPTS 环境变量是否设置正确，OpenOCD 根据此变量来查找-f 指定的文件，详见 安装 OpenOCD。此外，还需要检查配置文件是否确实位于该路径下。
• 如果出现 JTAG 错误（例如输出为 ...all ones 或 ...all zeroes）,请检查硬件连接是否正确，除了 ESP32 的管脚之外是否还有其他信号连接到了 JTAG，并查看是否所有器件都已经上电。

上传待调试的的应用程序

按照正常步骤构建并上传 ESP32 应用程序，具体请参阅 第五步：开始使用 ESP-IDF 吧 章节。

除此之外，您还可以使用 OpenOCD 通过 JTAG 接口将应用程序镜像烧写到 flash 中，命令如下:

```
openocd -f board/esp32-wrover-kit-3.3v.cfg -c "program_esp filename.bin 0x10000 ... verify exit"
```

其中 OpenOCD 的烧写命令 program_esp 格式如下:

```
program_esp <image_file> <offset> [verify] [reset] [exit]
```

• image_file - 程序镜像文件存放的路径
• offset - 镜像烧写到 flash 中的偏移地址
• verify - 烧写完成后校验 flash 中的内容（可选）
• reset - 烧写完成后重启目标（可选）
• exit - 烧写完成后退出 OpenOCD（可选）

现在可以调试应用程序了，请按照以下章节中的步骤进行操作。

### 4.17.6 启动调试器

ESP32 的工具链中带有 GNU 调试器（简称 GDB），它和其它工具链软件共同存放于 xtensa-esp32-elf-gdb 中。除了直接在命令行终端中调用并操作 GDB 外，也可以在 IDE（例如 Eclipse，Visual Studio Code 等）中进行调用。使用图形用户界面间接操作 GDB，这一方法无需在终端中输入任何命令。

关于以上两种调试器的使用方法，详见以下链接。

• 使用 Eclipse 调试
• 使用命令行调试

建议首先检查调试器能否在命令行终端下正常工作，然后再使用 Eclipse 集成开发环境 进行调试工作。
4.17.7 调试范例

本节适用于不熟悉 GDB 的用户，下文将使用 get-started/blink 下简单的应用程序来演示调试会话的工作流程，同时会介绍以下常用的调试操作:

1. 浏览代码，查看堆栈和线程
2. 设置和清除断点
3. 手动暂停目标
4. 单步执行代码
5. 查看并设置内存
6. 观察和设置程序变量
7. 设置条件断点

此外还会提供在命令终端下使用 GDB 调试的例子。

备注：调试 FreeRTOS 对象目前仅适用于命令行调试。

在演示之前，请完成 ESP32 目标板设置并加载 get-started/blink 至 ESP32 中。

4.17.8 从源码构建 OpenOCD

以下文档分别介绍了如何在各操作系统平台上从源码构建 OpenOCD。

Windows 环境下从源码编译 OpenOCD

备注：本文介绍了如何从 OpenOCD 源文件构建二进制文件。如果您想要更快速地构建，也可以从乐鑫 GitHub 直接下载 OpenOCD 的预构建二进制文件，而无需自己编译（详细信息，请参阅安装 OpenOCD）。

备注：本文涉及的命令行操作均在装有 MINGW32 子系统的 MSYS2 shell 环境中进行了验证。

安装依赖的软件包 安装编译 OpenOCD 所需的软件包：

```
pacman -S --noconfirm --needed autoconf automake git make \
mingw-w64-i686-gcc \
mingw-w64-i686-toolchain \
mingw-w64-i686-libtool \
mingw-w64-i686-pkg-config \
mingw-w64-cross-winthreads-git \
p7zip
```

下载 OpenOCD 源码 支持 ESP32 的 OpenOCD 源码可以从乐鑫官方 GitHub 获取，网址为 https://github.com/espressif/openocd-esp32。您可以在 Git 中使用以下命令来拉取源代码：

```
cd ~/esp

  git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码保存在 ~/esp/openocd-esp32 目录下。
Chapter 4. API 指南

下载 libusb 构建 OpenOCD 需使用 libusb 库。请执行以下命令来下载特定版本的 libusb，并将其解压至当前目录。

```
wget https://github.com/libusb/libusb/releases/download/v1.0.22/libusb-1.0.22.7z
7z x -olibusb ./libusb-1.0.22.7z
```

现在需要导入以下变量，以便将 libusb 库与 OpenOCD 构建相关联。

```
export CPPFLAGS="-I$(PWD)/libusb/include/libusb-1.0"
export LDLIBS="-L$(PWD)/libusb/MinGW32/.libs/dll"
```

构建 OpenOCD 配置和构建 OpenOCD，请参考以下命令：

```
cd ~/esp/openocd-esp32
export CPPFLAGS="-D__USE_MINGW_ANSI_STDIO=1 -Wno-error"
export CFLAGS="-Wno-error"
./bootstrap
./configure --disable-doxygen-pdf --enable-ftdi --enable-jlink --enable-ulink --build=i686-w64-mingw32 --host=i686-w64-mingw32
make
cp ../libusb/MinGW32/dll/libusb-1.0.dll ./src
cp /opt/i686-w64-mingw32/bin/libwinpthread-1.dll ./src
```

构建完成后，OpenOCD 的二进制文件将被保存于 ~/esp/openocd-esp32/src/ 目录下。

您可以使用 make install 将其复制到指定位置。

- 您可以在配置 OpenOCD 时指定这一位置，也可以在调用 make install 前设置 export DESTDIR="/custom/install/dir"。
- 如果您已经安装过其他开发平台的 OpenOCD，请跳过此步骤，否则原来的 OpenOCD 可能会被覆盖。

备注：

- 如果发生错误，请解决后再次尝试编译，直到 make 成功为止。
- 如果 OpenOCD 存在子模块问题，请 cd 到 openocd-esp32 目录，并输入 git submodule update --init 命令。
- 如果 ./configure 成功运行，JTAG 被使用的信息会被打印在 OpenOCD configuration summary 下面。
- 如果您的设备信息未显示在日志中，请根据 ./openocd-esp32/doc/INSTALL.txt 文中的描述使用 ./configure 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 openocd-esp32/README.Windows。
- 请记得将 libusb-1.0.dll 和 libwinpthread-1.dll 从 ~/esp/openocd-esp32/src 复制到 OOCD_INSTALLDIR/bin。

一旦 make 过程完成，OpenOCD 的可执行文件会被保存到 ~/esp/openocd-esp32/src/openocd 目录下。

完整编译过程 OpenOCD 编译过程中所调用的所有命令都已包含在以下代码片段中，您可以将其复制到 shell 脚本中，以便快速执行：

```
pacman -S --noconfirm --needed autoconf automake git make mingw-w64-i686-gcc mingw-w64-i686-toolchain mingw-w64-i686-libtool mingw-w64-i686-pkg-config mingw-w64-cross-winpthreads-gcc p7zip
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
wget https://github.com/libusb/libusb/releases/download/v1.0.22/libusb-1.0.22.7z
```

(下页继续)
7z x -olibusb ./libusb-1.0.22.7z
export CPPFLAGS="$CPPFLAGS -I$(PWD)/libusb/include/libusb-1.0"; export LDFLAGS="$LDFLAGS -L$(PWD)/libusb/MinGW32/.libs/dll"

export CPPFLAGS="SCPPFLAGS -D__USE_MINGW_ANSI_STDIO=1 -Wno-error"; export CFLAGS="-Wno-error";

cd ~/esp/openocd-esp32
./bootstrap
./configure --disable-doxygen-pdf --enable-ftdi --enable-jlink --enable-ulink --build-i686-w64-mingw32 --host-i686-w64-mingw32
make

cp .. /libusb/MinGW32/dll/libusb-1.0.dll ./src

cp /opt/i686-w64-mingw32/bin/libwinpthread-1.dll ./src

# # optional
# export DESTDIR="$PWD"
# make install
# cp ./src/libusb-1.0.dll $DESTDIR/mingw32/bin
# cp ./src/libwinpthread-1.dll $DESTDIR/mingw32/bin

下一步 想要进一步配置调试环境，请前往配置 ESP32 目标板 章节。

Linux 环境下从源码编译 OpenOCD

除了从 Espressif 官方 直接下载 OpenOCD 可执行文件，你还可以选择从源码编译得到 OpenOCD。如果想要快速设置 OpenOCD 而不是自行编译，请备份好当前文件，前往 安装 OpenOCD 章节查阅。

下载 OpenOCD 源码 支持 ESP32 的 OpenOCD 源代码可以从乐鑫官方的 GitHub 获得，网址为 https://github.com/espressif/openocd-esp32，请使用以下命令来下载源代码：

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码被保存在 ~/esp/openocd-esp32 目录中。

安装依赖的软件包 安装编译 OpenOCD 所需的软件包。

备注：依次安装以下软件包，检查安装是否成功，然后继续下一个软件包的安装。在进行下一步操作之前，要先解决当前报告的问题。

```
sudo apt-get install make
sudo apt-get install libtool
sudo apt-get install pkg-config
sudo apt-get install autoconf
sudo apt-get install automake
sudo apt-get install texinfo
sudo apt-get install libusb-1.0
```

备注：
- pkg-config 应为 0.2.3 或以上的版本。
- autoconf 应为 2.6.4 或以上的版本。
- automake 应为 1.9 或以上的版本。
Chapter 4. API

• 当使用 USB-Blaster，ASIX Presto，OpenJTAG 和 FT2232 作为适配器时，需要下载安装 libFTDI 和 FTD2XX 的驱动。
• 当使用 CMSIS-DAP 时，需要安装 HIDAPI。

构建 OpenOCD 配置和构建 OpenOCD 的流程如下:

```
cd ~/esp/openocd-esp32
./bootstrap
./configure
make
```

你可以选择最后再执行 sudo make install，如果你已经安装过别的开发平台的 OpenOCD，请跳过这个步骤，因为它可能会覆盖掉原来的 OpenOCD。

备注:
• 如果发生错误，请解决后再次尝试编译，直到 make 成功为止。
• 如果 OpenOCD 存在于模块问题，请 cd 到 openocd-esp32 目录，并输入 git submodule update --init 命令。
• 如果 ./configure 成功运行，JTAG 被使能的信息会被打印在 OpenOCD configuration summary 下面。
• 如果您的设备信息未显示在日志中，请根据 ..openocd-esp32/doc/INSTALL.txt 文中的描述使用 ./configure 启用它。
• 有关编译 OpenOCD 的详细信息，请参阅 openocd-esp32/README。

一旦 make 过程成功结束，OpenOCD 的可执行文件会被保存到 ~/openocd-esp32/bin 目录中。

下一步 想要进一步配置调试环境，请前往配置 ESP32 目标板 章节。

MacOS 环境下从源码编译 OpenOCD

除了从 Espressif 官方 直接下载 OpenOCD 可执行文件，你还可以选择从源码编译得到 OpenOCD。如果想要快速设置 OpenOCD 而不是自行编译，请备份好当前文件，前往安装 OpenOCD 章节查阅。

下载 OpenOCD 源码 支持 ESP32 的 OpenOCD 源代码可以从乐鑫官方的 GitHub 获取。网址为 https://github.com/espressif/openocd-esp32。请使用以下命令来下载源代码:

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码被保存在 ~/esp/openocd-esp32 目录中。

安装依赖的软件包 使用 Homebrew 安装编译 OpenOCD 所需的软件包:

```
brew install automake libtool libusb wget gcc@4.9 pkg-config
```

构建 OpenOCD 配置和构建 OpenOCD 的流程如下:

```
cd ~/esp/openocd-esp32
./bootstrap
./configure
make
```
你可以选择最后再执行 `sudo make install`。如果你已经安装过别的开发平台的 OpenOCD，请跳过这个步骤。因为它可能会覆盖掉原来的 OpenOCD。

### 备注:
- 如果发生错误，请解决后再次尝试编译，直到 make 成功为止。
- 发生 `Unknown command 'raggedright'` 错误可能是因为安装的 texinfo 版本不对，或是由于没有将其添加到 PATH 路径。为了解决该问题，在运行 ./bootstrap 前，请先运行如下命令确保安装合适版本的 texinfo 并将其添加到 PATH 路径:

```bash
brew install texinfo
export PATH=/usr/local/opt/texinfo/bin:$PATH
```
- 如果 OpenOCD 存在子模块问题，请 cd 到 openocd-esp32 目录，并输人 git submodule update --init 命令。
- 如果 ./configure 成功运行，JTAG 被使能的信息会被打印在 OpenOCD configuration summary 下面。
- 如果您的设备信息未显示在日志中，请根据 ./openocd-esp32/doc/INSTALL.txt 文中的描述使用 ./configure 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 openocd-esp32/README.OSX。

一旦 make 过程成功结束，OpenOCD 的可执行文件会被保存到 ~/openocd-esp32/src/openocd 目录中。

#### 下一步
想要进一步配置调试环境，请前往 配置 ESP32 目标板 章节。

本文档在演示中所使用的 OpenOCD 是预编译好的。二进制文件的版本，安装 OpenOCD 章节中有所介绍。

如果要使用本地源代码编译的 OpenOCD 程序，需要将相应可执行文件的路径修改为 src/openocd，并设置 OPENOCD_SCRIPTS 环境变量，使得 OpenOCD 能够找到配置文件。Linux 和 macOS 用户可以执行:

```bash
cd ~/openocd-esp32
export OPENOCD_SCRIPTS=$PWD/tcl
```

Windows 用户可以执行:

```bash
cd %USERPROFILE%\esp\openocd-esp32
set "OPENOCD_SCRIPTS=%CD%\tcl"
```

针对 Linux 和 macOS 用户，运行本地编译的 OpenOCD 的示例:

```bash
src/openocd -f board/esp32-wrover-kit-3.3v.cfg
```

Windows 用户的示例如下:

```bash
src\openocd -f board\esp32-wrover-kit-3.3v.cfg
```

### 4.17.9 注意事项和补充内容

本节列出了上文中提到的所有注意事项和补充内容的链接。

#### 注意事项和补充内容

本节提供了本指南中各部分提到的一些注意事项和补充内容。
关于断点的补充知识 使用常规的 GDB 命令 `hb myFunction` 给某个函数设置硬件断点时，如果该函数位于内存中，并且此时还有可用的硬件断点，那么 GDB 就会使用 32 个硬件断点，否则就会使用 32 个软件内存断点中的一个来模拟。这个规则同样适用于 b myFunction 之类的命令。在这种情况 GDB 会自己决定该使用哪种类型的断点。如果 myFunction 位于可写区域 (IRAM)，那就会使用软件 IRAM 断点，否则就会像处理 hb 命令一样使用硬件断点或者软件内存断点。

**闪光映射 vs 软件内存断点** 为了在内存中设置或者清除软件断点，OpenOCD 需要知道它们在内存中的地址。为了完成从 ESP32 的地址空间到内存地址的转换，OpenOCD 使用内存中程序代码区域的映射。这些映射被保存在程序映像的头部，位于二进制数据 (代码段和数据段) 之前，并且用于内存中程序的映射。为了支持软件内存断点，OpenOCD 需要知道程序映像在内存中的位置。默认情况下，OpenOCD 会在 0x8000 处读取内存映像并使用第一个找到的应用程序映像的映射，但是也可能会存在无法工作的代码，比如内存不在标准的内存位置，甚至可能有多个映像：一个出厂映像和一个 OTA 映像。你可能想要调试其中的任意一个。为了涵盖所有可能的调试情况，OpenOCD 支持特殊的命令，用于指定待调试的应用程序映像在内存中的具体位置。该命令具有以下格式：

```plain
esp appimage_offset <offset>
```

偏移量应为十六进制格式，如果要恢复默认行为，可以将偏移地址设置为 -1。

**备注**：由于 GDB 在连接 OpenOCD 时仅仅请求一次内存映射，所以可以在 TCL 配置文件中指定该命令，或者通过命令行传递给 OpenOCD。对于后者，命令行示例如下：

```bash
openocd -f board/esp32-wrover-kit-3.3v.cfg -c "init; halt; esp appimage_offset -0x210000"
```

另外还可以通过 OpenOCD 的 telnet 会话执行该命令，然后再连接 GDB，不过这种方式似乎没有那么便捷。

**“next” 命令无法跳过子程序的原因** 当使用 `next` 命令单步执行代码时，GDB 会在子程序的前面设置一个断点 (两个中可用的一个)，这样就可以跳过进入子程序内部的细节。如果这两个断点已经用在代码的其它位置，那么 `next` 命令将不起作用。在这种情况下，请删掉一个断点以使其中一个变得可用。当两个断点已经被使用时，`next` 命令会像 `step` 命令一样工作，调试器就会进入子程序内部。

**OpenOCD 支持的编译时的选项** ESP-IDF 有一些针对 OpenOCD 调试功能的选项可以在编译时进行设置：

- `CONFIG_ESP_DEBUG_OCDAWARE` 默认会被使能。如果程序抛出了不可修复或者未处理的异常，并且此时已经连接上了 JTAG 调试器 (即 OpenOCD 正在运行)，那么 ESP-IDF 将会进入调试器工作模式。
- `CONFIG_FREERTOSWATCHPOINT_END_OF_STACK` 默认没有使能。在所有任务堆栈的末尾设置观察点，从 1 号开始索引，这是调试任务堆栈溢出的最准确的方式。

更多有关设置编译时的选项的信息，请参阅 [项目配置菜单](#)。
支持 FreeRTOS OpenOCD 完全支持 ESP-IDF 自带的 FreeRTOS 操作系统，GDB 会将 FreeRTOS 中的任务当做线程。使用 GDB 命令 i threads 可以查看所有的线程，使用命令 thread n 可以切换到某个具体任务的堆栈，其中 n 是线程的编号。检测 FreeRTOS 的功能可以在配置目标时被禁用。更多详细信息，请参阅根据目标芯片配置 OpenOCD。

GDB 具有 FreeRTOS 支持的 Python 扩展模块。在系统要求满足的情况下，通过 idf.py gdb 命令，ESP-IDF 会将该模块自动加载到 GDB 中。详细信息请参考调用 FreeRTOS 对象。

在 OpenOCD 的配置文件中设置 SPI 闪存的工作电压 ESP32 的 MTD 管脚是用于 JTAG 通信的四个管脚之一，同时也是 ESP32 的 bootstrapping 管脚。上电时，ESP32 会在 MTD 管脚上采样二进制电平，据此来设置内部的稳压器，用于给外部的 SPI 闪存芯片供电。如果上电时 MTD 管脚上的二进制电平为低电平，则稳压器会被设置为 3.3 V。如果 MTD 管脚为高电平，则稳压器会被设置为 1.8 V。MTD 管脚通常需要一个上拉电阻或者直接使用内部的弱下拉电阻（详见 ESP32 系列芯片技术规格书），具体取决于所使用的 SPI 芯片的类型。但是一旦连接上 JTAG 后，原来用于实现 bootstrapping 功能的上拉或者下拉电阻都会被覆盖掉。

为了解决这个问题，OpenOCD 的板级配置文件（例如 ESP-WROVER-KIT 开发板的 board\exp32-wrover-kit-3.3v.cfg）提供了 ESP32_FLASH_VOLTAGE 参数来设置 TDO 信号线在空闲状态下的二进制电平，这样就可以减少由于闪存电压不正确而导致的应用程序启动不良的几率。

查看 JTAG 连接的 ESP32 模块的规格书，检查其 SPI 闪存芯片的供电电压值，然后再相应的设置 ESP32_FLASH_VOLTAGE。大多数 WROOM 模块使用 3.3 V 的闪存芯片。早于 ESP32-WROVER-B 的 WROVER 模块使用 1.8 V 闪存芯片，而 ESP32-WROVER-B 和 E 模块使用 3.3 V 闪存芯片。

优化 JTAG 的速度 为了实现更高的数据通信速率同时最小化丢包数，建议优化 JTAG 时钟频率的设置，使其达到 JTAG 能稳定运行的最大值。为此，请参考以下建议。

1. 如果 CPU 以 80 MHz 运行，则 JTAG 时钟频率的上限为 20 MHz；如果 CPU 以 160 MHz 或者 240 MHz 运行，则上限为 26 MHz。
2. 根据特定的 JTAG 适配器和连接线缆的长度，你可能需要将 JTAG 的工作频率降低至 20 / 26 MHz 以下。
3. 在某些特殊情况下，如果你看到 DSR/DIR 错误（并且它并不是由 OpenOCD 试图从一个没有物理存储器映射的地址空间读取数据而导致的），请降低 JTAG 的工作频率。
4. ESP-WROVER-KIT 能够稳定运行在 20 / 26 MHz 频率下。

调试器的启动命令的含义 在启动时，调试器发出一系列命令来复位芯片并使其在特定的代码行停止运行。这个命令序列（如下所示）支持自定义，用户可以选择在最方便合适的代码行开始调试工作。

- set remote hardware-watchpoint-limit 2 —限制 GDB 只使用 ESP32 支持的两个硬件观察点。更多详细信息，请查阅 GDB 配置远程目标。
- mon reset halt —复位芯片并使 CPU 停止运行。
- flushregs —mon 命令无法通知 GDB 目标状态已经更改，GDB 会假设在 mon reset halt 之前所有的任务堆栈仍然有效。实际上，复位后目标状态将发生变化。执行 flushreg 是一种强制 GDB 从目标获取最新状态的方法。
- thb app_main —在 app_main 处插入一个临时的硬件断点，如果有需要，可以将其替换为其他函数名。
- c —恢复程序运行，它将会在 app_main 的断点处停止运行。

根据目标芯片配置 OpenOCD OpenOCD 有很多种配置文件 (*.cfg)，它们位于 OpenOCD 安装目录的 share/openocd/scripts 子目录中（或者在 OpenOCD 源码目录的 tcl/scripts 目录中）。本文主要介绍 board, interface 和 target 这三个目录。

- interface 包含了例如 ESPProg、J-Link 这些 JTAG 适配器的配置文件。
- target 包含了目标芯片或者模组的配置文件。
- board 包含有关于 JTAG 适配器的开发板的配置文件，这些配置文件会根据实际的 JTAG 适配器和芯片/模组来导出某个具体的 interface 和 target 的配置。

ESP32 可以使用的配置文件如下表所示：

<table>
<thead>
<tr>
<th>ESP32-WROVER-B</th>
<th>20 / 26 MHz</th>
<th>3.3 V</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32-WROVER-B</td>
<td>1.8 V</td>
<td></td>
</tr>
</tbody>
</table>

注意：ESP32-WROVER-B 模块使用 3.3 V 闪存芯片。早于 ESP32-WROVER-B 的 WROVER 模块使用 1.8 V 闪存芯片，而 ESP32-WROVER-B 和 E 模块使用 3.3 V 闪存芯片。
### 表 24: ESP32 相关的 OpenOCD 配置文件

<table>
<thead>
<tr>
<th>名字</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>board/esp32-wrover-kit-3.3v.cfg</td>
<td>板载 3.3 V 模组 (ESP32-WROOM-32, ESP32-WROVER-B, ESP32-WROVER-E) 的 ESP-WROVER-KIT 开发板配置文件</td>
</tr>
<tr>
<td>board/esp32-wrover-kit-1.8v.cfg</td>
<td>板载 1.8 V 模组 (ESP32-WROVER) 的 ESP-WROVER-KIT 开发板配置文件</td>
</tr>
<tr>
<td>board/esp32-ethernet-kit-3v.cfg</td>
<td>板载 3.3 V 模组 (ESP32-WROVER-B/ESP32-WROVER-E) 的 ESP-Ethernet-KIT 开发板配置文件</td>
</tr>
<tr>
<td>target/esp32.cfg</td>
<td>ESP32 的目标配置文件，可以和某个 interface/ 下的配置文件一同使用</td>
</tr>
<tr>
<td>target/esp32-solo-1.cfg</td>
<td>ESP32-SOLO-1 模组的目标配置文件，和 esp32.cfg 的差别在于它可以配置一个 CPU</td>
</tr>
<tr>
<td>interface/ftdi/esp32_devkitj_v1.cfg</td>
<td>适用于 ESP-WROVER-KIT 和 ESP-Prog 的 JTAG 适配器配置文件</td>
</tr>
</tbody>
</table>

如果你使用的开发板已经有一份预定义好的配置文件，你只需将该文件通过 -f 参数告诉 OpenOCD。如果你的开发板不在上述列表中，你需要使用多个 -f 参数来告诉 OpenOCD 你选择的 interface 和 target 配置文件。

#### 自定义配置文件
OpenOCD 的配置文件是用 TCL 语言编写的，包含了定制和编写脚本的各种选项。这在非标准调试的场景中非常有用，更多关于 TCL 脚本的内容请参考 OpenOCD 参考手册。

**OpenOCD 中的配置变量**
你还可以视情况在导入 target 配置文件之前，设定如下变量的值。可以在自定义配置文件中，或者通过命令行传递。

TCL 语言中为变量赋值的语法是:

```tcl
set VARIABLE_NAME value
```

在命令行中为变量赋值请参考如下示例（请把 .cfg 配置文件替换成你自己的开发板配置）:

```bash
openocd -c 'set VARIABLE_NAME value' -f board/esp-xxxxx-kit.cfg
```

请记住，一定要在导入配置文件之前设置这些变量，否则变量的值将不会生效。为多个变量赋值需要重复多次 -c 选项。

### 表 25: 通用的 ESP 相关的 OpenOCD 变量

<table>
<thead>
<tr>
<th>变量名</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_RTOS</td>
<td>设置成 none 可以关闭 OpenOCD 对 RTOS 的支持，这样的话，你将无法在 GDB 中查看到线程列表，这个功能在调试 FreeRTOS 本身的时候会很有用，可以单步调试调度器的代码。</td>
</tr>
<tr>
<td>ESP_FLASH_SIZE</td>
<td>设置成 0 可以关闭对 flash 断点的支持。</td>
</tr>
<tr>
<td>ESP_SEMIHOST_BASEDIR</td>
<td>设置 semihosting 在主机端的默认目录。</td>
</tr>
</tbody>
</table>

### 表 26: ESP32 相关的 OpenOCD 变量

<table>
<thead>
<tr>
<th>名字</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32_FLASH_VOLTAGE</td>
<td>如果 ESP32 模组集成的是 1.8 V 的 flash，将该变量设置为 1.8，详情请参考 OpenOCD 的配置文件中设置 SPI 闪存的工作电压</td>
</tr>
<tr>
<td>ESP32_ONLYCPU</td>
<td>对于多核芯片，将该值设置为 1 可以仅启用单核调试功能</td>
</tr>
</tbody>
</table>
复位 ESP32 通过在 GDB 中输入 `mon reset` 或者 `mon reset halt` 来复位板子。

不要将 JTAG 管脚用于其他功能 如果除了 ESP32 模组和 JTAG 适配器之外的其他硬件也连接到了 JTAG 管脚，那么 JTAG 的操作可能会受到干扰。ESP32 JTAG 使用以下管脚：

<table>
<thead>
<tr>
<th>ESP32 管脚</th>
<th>JTAG 信号</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTD0/GPIO15</td>
<td>TDO</td>
</tr>
<tr>
<td>MTD1/GPIO12</td>
<td>TDI</td>
</tr>
<tr>
<td>MTCK/GPIO13</td>
<td>TCK</td>
</tr>
<tr>
<td>MTMS/GPIO14</td>
<td>TMS</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>

如果用户应用程序更改了 JTAG 管脚的配置，JTAG 通信可能会失败。如果 OpenOCD 正确初始化（检测到两个 Tensilica 内核），但在程序运行期间失去了同步并报出大量 DTR/DIR 错误，则应用程序可能将 JTAG 管脚重新配置为其他功能或者用户忘记将 Vtsr 连接到 JTAG 适配器。

下面是 GDB 在应用程序进入重新配置 MTDO/GPIO15 作为输入代码后报告的一系列错误摘要:

```
cpu0: xtnsa_resume (line 431): DSR (FFFFFFFF) indicates target still busy!
cpu0: xtnsa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
  an exception!
cpu0: xtnsa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
  an overrun!
cpu1: xtnsa_resume (line 431): DSR (FFFFFFFF) indicates target still busy!
cpu1: xtnsa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
  an exception!
cpu1: xtnsa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
  an overrun!
```

JTAG 与闪存加密和安全引导 默认情况下，开启了闪存加密和（或者）安全引导后，系统在首次启动时，引导程序会烧写 eFuse 的某个比特，从而将 JTAG 永久关闭。

Kconfig 配置项 `CONFIG_SECURE_BOOT_ALLOW_JTAG` 可以改变这个默认行为，使得用户即使开启了安全引导或者闪存加密，仍会保留 JTAG 的功能。

然而，因为设置软件断点的需要，OpenOCD 会尝试自动读取 flash 中的内容，这会带来两个问题:

- 软件断点和闪存加密是不兼容的，目前 OpenOCD 尚不支持对 flash 中的内容进行加密和解密。
- 如果开启了安全引导功能，设置软件断点会改变被签名的程序的摘要，从而使得签名失效。这也意味着，如果设置了软件断点，系统会在下次重启时的签名验证阶段失败，导致无法启动。

关闭 JTAG 的软件断点功能，可以在启动 OpenOCD 时在命令行额外加一项配置参数 `-c 'set ESP_FLASH_SIZE 0'`，请参考 `OpenOCD` 中的配置变量。

备注：同样地，当启用该选项，并且在调试过程中设置了软件断点，引导程序将无法校验通过应用程序的签名。

JTAG 和 ESP32-WROOM-32 AT 固件兼容性问题 ESP32-WROOM 系列模块预装了 AT 固件。该固件将 GPIO12 至 GPIO15 管脚配置为 SPI 从属接口，使得无法使用 JTAG。

要想使用 JTAG，需要编译新的固件，新的固件不能使用专门用于 JTAG 通信的管脚（GPIO12 至 GPIO15），然后将固件烧录到模组中，请参考不要将 JTAG 管脚用于其他功能。
报告 OpenOCD / GDB 的问题 如果你遇到 OpenOCD 或者 GDB 程序本身的问题，并且在网上没有找到可用的解决方案，请前往 https://github.com/espressif/openocd-esp32/issues 新建一个议题。

1. 请在问题报告中提供你使用的配置的详细信息:
   a. JTAG 适配器类型。
   b. 用于编译和加载正在调试的应用程序的 ESP-IDF 版本号。
   c. 用于调试的操作系统的详细信息。
   d. 操作系统是否在本地计算机运行还是在虚拟机上运行？
2. 创建一个能够演示问题的示例示例工程，描述复现该问题的步骤。且这个调试示例不能受到 Wi-Fi 协议栈引入的非确定性行为的影响，这样复现问题时，更容易复现。
3. 在启动命令中添加额外的参数来自定义调试日志。

   **OpenOCD 端**:
   ```
   openocd -l openocd_log.txt -d3 -f board/esp32-wrover-kit-3.3v.cfg
   ```

   这种方式会将日志输出到文件，但是它会阻止调试信息打印在终端上，当有大量信息需要输出的时候（比如调试等级提高到 -d3）这是个不错的选择。如果你仍然希望在屏幕上看到调试日志，请改用以下命令:
   ```
   openocd -d3 -f board/esp32-wrover-kit-3.3v.cfg 2>&1 | tee openocd.log
   ```

   **GDB 端**:
   ```
   xtensa-esp32-elf-gdb -ex "set remotelogfile gdb_log.txt" <all other options>
   ```

   也可以将命令 remotelogfile gdb_log.txt 添加到 gdbinit 文件中。
4. 请将 openocd_log.txt 和 gdb_log.txt 文件附在你的问题报告中。

### 4.17.10 相关文档

**使用调试器**

本节介绍以下几种配置和运行调试器的方法:

- 使用 Eclipse 调试
- 使用命令行调试
- 使用 idf.py 进行调试

**使用 Eclipse 调试**

**备注**: 建议您首先通过 `idf.py` 或命令行 检查调试器是否正常工作，然后再转到使用 Eclipse 平台。

标准的 Eclipse 安装流程默认安装调试功能，另外您还可以使用插件来调试，比如“GDB Hardware Debugging”。这个插件用起来非常方便，本指南会详细介绍该插件的使用方法。

首先，打开 Eclipse 并转到“Help” > “Install New Software” 来安装“GDB Hardware Debugging”插件。

安装完成后，按照以下步骤配置调试会话。请注意，一些配置参数是通用的，有些则针对特定项目。我们会通过配置“blink”示例项目的调试环境来进行展示，请参照 Eclipse Plugin 介绍的方法将该示例项目添加到 Eclipse 的工作空间。示例项目 get-started/blink 的源代码可以在 ESP-IDF 仓库的 examples 目录下找到。

1. 在 Eclipse 中，进入 Run > Debug Configuration，会出现一个新的窗口。在窗口的左侧窗格中，双击“GDB Hardware Debugging”(或者选择“GDB Hardware Debugging”然后按下“New”按钮)来新建一个配置。
2. 在右边显示的表单中，“Name:”一栏中输入配置的名称，例如：“Blink checking”。
3. 在下面的“Main”选项卡中，点击“Project”边上的“Browse”按钮，然后选择当前的“blink”项目。
4. 在下一行的“C/C++ Application:”中，点击“Browse”按钮，选择“blink.elf”文件。如果“blink.elf”文件不存在，那么有可能该项目还没有编译，请参考 Eclipse Plugin 指南中的介绍。
5. 在“Build (if required) before launching”下面点击“Disable auto build”。

Espressif Systems 2272 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
上述步骤 1 - 5 的示例输入如下图所示。

图 37: GDB 硬件调试的配置 - Main 选项卡

6. 点击 “Debugger” 选项卡，在 “GDB Command” 栏中输入 xtensa-esp32-elf-gdb 来调用调试器。
7. 更改 “Remote host”的默认配置，在 “Port number” 下面输入 3333。
   上述步骤 6 - 7 的示例输入如下图所示。
8. 最后一个需要更改默认配置的选项是 “Startup” 选项卡。在 “Initialization Commands” 下，取消选中 “Reset and Delay (seconds)” 和 “Halt”，然后在下面一栏中输入以下命令:

   ```
   mon reset halt
   flushregs
   set remote hardware-watchpoint-limit 2
   ```

备注: 如果您想在启动新的调试会话之前自动更新闪存中的镜像，请在 “Initialization Commands” 文本框的开头添加以下命令行:

   ```
   mon reset halt
   mon program_esp $\{workspace_loc:blink/build/blink.bin\} 0x10000 verify
   ```

有关 program_esp 命令的说明请参考上传待调试的应用程序 章节。
9. 在 “Load Image and Symbols” 下，取消选中 “Load image” 选项。
10. 在同一个选项卡中继续往下浏览，建立一个初始断点用来在调试器复位后暂停 CPU。插件会根据 “Set break point at:” 一栏中输入的函数名，在该函数的开头设置断点。选中这一选项，并在相应的字段中输入 app_main。
11. 选中 “Resume” 选项，这会使得程序在每次调用步骤 8 中的 mon reset halt 后恢复，然后在 app_main 的断点处停止。
   上述步骤 8 - 11 的示例输入如下图所示。
图 38: GDB 硬件调试的配置 - Debugger 选项卡
图 39: GDB 硬件调试的配置 - Startup 选项卡
上面的启动序列看起来有些复杂，如果您对其中的初始化命令不太熟悉，请查阅调试器的启动命令的含义章节获取更多说明。

12. 如果您前面已经完成配置 ESP32 目标板中介绍的步骤，那么目标正在运行并准备与调试器进行对话。按下“Debug”按钮就可以直接调试。否则请按下“Apply”按钮保存配置，返回配置 ESP32 目标板章节进行配置，最后再回到这里开始调试。

一旦所有 1-12 的配置步骤都已经完成，Eclipse 就会打开“Debug”视图，如下图所示。

![Eclipse 中的调试视图](image)

如果您不太了解 GDB 的常用方法，请查阅使用 Eclipse 的调试示例文章中的调试示例章节调试范例。

### 使用命令行调试

1. 为了能够启动调试会话，需要先启动并运行目标，如果还没有完成，请按照配置 ESP32 目标板中的介绍进行操作。
2. 打开一个新的终端会话并前往待调试的项目目录，比如:
   ```bash
cd ~/esp/blink
   ```
3. 当启动调试器时，通常需要提供几个配置参数和命令。为了避免每次都在命令行中逐行输入这些命令，您可以新建一个配置文件，并将其命名为 gdbinit:
   ```bash
target remote :3333
set remote hardware-watchpoint-limit 2
mon reset halt
flushregs
thb app_main
c
```

将此文件保存在当前目录中。
有关 gdbinit 文件内部的更多详细信息，请参阅调试器的启动命令的含义 章节。

4. 准备好启动 GDB，请在终端中输入以下内容:

```bash
tenxsa-esp32-elf-gdb -x gdbinit build/blink.elf
```

5. 如果前面的步骤已经正确完成，您会看到如下所示的输出日志，在日志的最后会出现 (gdb) 提示符:

```bash
user-name@computer-name:~/esp/blink$ xtensa-esp32-elf-gdb -x gdbinit build/blink.elf
GDB (crosstool-NG crosstool-ng-1.22.0-61-gab8375a) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-build_pc-linux-gnu --target=xtensa-
esp32-elf".
Type *show configuration* for configuration details.
For bug reporting instructions, please see:
Find the GDB manual and other documentation resources online at:
For help, type "help".
Type *apropos word* to search for commands related to "word"...
Reading symbols from build/blink.elf...done.
0x400d10d8 in esp_vApplicationIdleHook () at /home/user-name/esp/esp-idf/
components/esp32./freertos_hooks.c:52
52 asm("waiti 0");
JTAG tap: esp32.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),
--part: 0x2003, ver: 0x1)
JTAG tap: esp32.slave tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),
--part: 0x2003, ver: 0x1)
esp32: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
esp32: Core was reset (pwrstat=0x5F, after clear 0x0F).
esp32 halted. PRO_CPU: PC=0x50000000 (active)  APP_CPU: PC=0x00000000
esp32: target state: halted
esp32: Core was reset (pwrstat=0x1F, after clear 0x0F).
Target halted. PRO_CPU: PC=0x40000400 (active)  APP_CPU: PC=0x40000400
esp32: target state: halted
Hardware assisted breakpoint 1 at 0x400db717: file /home/user-name/esp/blink/
--main./blink.c, line 43.
0x0: 0x00000000
Target halted. PRO_CPU: PC=0x400DB717 (active)  APP_CPU: PC=0x400D10D8
[New Thread 1073428656]
[New Thread 1073413708]
[New Thread 1073431316]
[New Thread 1073410672]
[New Thread 1073408876]
[New Thread 1073432196]
[New Thread 1073411552]
[Switching to Thread 1073411996]
Temporary breakpoint 1, app_main () at /home/user-name/esp/blink/main/.//blink.
--c:43
43 xTaskCreate(&blink_task, "blink_task", 512, NULL, 5, NULL);
(gdb)
```

注意上面日志的倒数第三行显示了调试器已经在 app_main() 函数的断点处停止，该断点在 gdbinit 文件中设定。由于处理器已经暂停运行，LED 也不会闪烁。如果这也是您看到的现象，您可以开始调试了。

如果您不太了解 GDB 的常用方法，请查阅使用命令行的调试示例 文章中的调试示例章节调试范例。
使用 idf.py 进行调试 您还可以使用 idf.py 更方便地执行上述提到的调试命令，可以使用以下命令：

1. idf.py openocd
   在终端中运行 OpenOCD，其配置信息来源于环境变量或者命令行，默认会使用 OPENOCD_SCRIPTS 环境变量中指定的脚本路径。如果你在 ESP-IDF 项目仓库中没有指定环境变量，那么命令中的路径会被覆盖这个变量的值。在启动开发板的 JTAG 配置，请使用环境变量 OPENOCD_COMMANDS 或命令行参数 --openocd-commands。如果这两者都没有被定义，那么 OpenOCD 会使用 --f board/esp32-wrover-kit-3.3v.config 参数来启动。
   2. idf.py gdb
      根据当前项目的 elf 文件自动导出 GDB 启动脚本，然后会按照使用命令行调试 中所描述的步骤启动 GDB。
   3. idf.py gdbtui
      启动 gdbtui，它在浏览器中打开调试器的前端界面。请在运行安装脚本时添加 “--enable-gdbtui” 参数，即运行 install.sh --enable-gdbtui。可以使用 gdbtui 参数来启动。
      如上这些命令也可以合并到一起使用。idf.py 会自动将后台进程（比如 openocd）最先运行，交互式进程中启动 GDB，gdbgui 最后运行。
      常用的组合命令如下所示：

```bash
idf.py openocd gdbgui monitor
```

上述命令会将 OpenOCD 运行至后台，然后启动 gdbgui 打开一个浏览器窗口，显示调试器的前端界面，最后在活动终端打开串口监视器。

调试示例

本节将介绍如何在 Eclipse 和命令行 中使用 GDB 进行调试的示例。

使用 Eclipse 的调试示例 请检查目标板是否已经准备好，并加载了 get-started/blink 示例代码。然后按照使用 Eclipse 调试 中介绍的步骤配置和启动调试器，最后选择让应用程序在 app_main() 建立的断点处停止。

本小节的示例

1. 浏览代码、查看堆栈和线程
2. 设置和清除断点
3. 手动暂停目标
4. 单步执行代码
5. 查看并设置内存
6. 观察和设置程序变量
7. 设置条件断点

浏览代码、查看堆栈和线程 当目标暂停时，调试器会在 “Debug” 窗口中显示线程的列表，程序暂停的代码行在下面的另一个窗口中被高亮显示，如下图所示。此时板子上的 LED 停止了闪烁。

暂停的程序所在线程也会被展开，显示函数调用的堆栈，它表示直到目标暂停所在代码行（下图高亮处）为止的所有函数的调用关系。1 号线程下函数调用堆栈的第一行包含了最后一个调用的函数 app_main()，根据下一行显示，它又是在函数 main_task() 中调用的。堆栈的每一行还包含调用函数的文件名和行号。通过单击每个堆栈的条目，在下面的窗口中，你将看到此文件的内容。

通过展开线程，你可以浏览整个应用程序。展开 5 号线程，它包含了更长的函数调用堆栈，你可以看到函数调用旁边的数字，比如 0x4000000c，它们代表未以源码形式提供的二进制代码所在的内存地址。

无论项目是以源代码还是仅以二进制形式提供，在右边一个窗口中，都可以看到反汇编后的机器代码。
图 41: Eclipse 中的 Debug 视图
图 42: 调试时目标停止
图 43: 浏览函数调用堆栈
回到1号线程中的`app_main()`函数所在的`blink.c`源文件。下面的示例将会以该文件为例介绍调试的常用功能。调试器可以轻松浏览整个应用程序的代码，这给单步调试代码和设置断点带来了很大的便利，下面将一一展开讨论。

### 设置和清除断点
在调试时，我们希望能够将关键的代码行停止应用程序，然后检查特定的变量、内存、寄存器和外设的状态。为此我们需要使用断点，以便在特定表行代码处快速访问和停止应用程序。

我们在控制LED状态发生变化的两处代码分别设置一个断点。基于以上代码列表，这两处分别为第33和36代码行。按住键盘上的“Control”键，双击`blink.c`文件中的行号33，并在弹出的对话框中点击“OK”按钮进行确定。如果不想想到此对话框，双击行号即可。执行同样操作，在第36行设置另外一个断点。

![设置和清除断点](image)

图44: 设置断点

断点的数量和位置信息会显示在右上角的“断点”窗口中。单击“Show Breakpoints Supported by Selected Target”图标可以刷新此列表。除了刚才设置的两个断点外，列表中可能还包含在调试器启动时设置在`app_main()`函数处的临时断点。由于最多只允许设置两个断点（详细信息请参阅可用的断点和观察点），你需要将其删除，否则调试会失败。

单击“Resume”（如果“Resume”按钮是灰色的，请先单击8号线程的`blink_task()`函数）后处理器将开始继续运行，并在断点处停止。再次单击“Resume”按钮，使程序再次运行，然后停在第二个断点处，依次类推。

每次单击“Resume”按钮恢复程序运行后，都会看到LED切换状态。

更多关于断点的信息，请参阅可用的断点和观察点和关于断点的补充知识。

### 手动暂停目标
在调试时，你可以恢复程序运行并输入代码等待某个事件发生或者保持无限循环而不设置任何断点。后者，如果想要返回调试模式，可以通过单击“Suspend”按钮来手动中断程序的运行。
图 45: 设置了三个断点 / 最多允许两个断点
在此之前，请删除所有的断点，然后单击“Resume”按钮。接着单击“Suspend”按钮，应用程序会停止在某个随机的位置，此时 LED 也将停止闪烁。调试器将展开线程并高亮显示停止的代码行。

图 46: 手动暂停目标

在上图所示的情况中，应用程序已经在 freertos_hooks.c 文件的第 52 行暂停运行，现在你可以通过单击“Resume”按钮再次将其恢复运行或者进行下面要介绍的调试工作。

单步执行代码 我们还可以使用“Step Into (F5)”和“Step Over (F6)”命令单步执行代码。这两者之间的区别是执行“Step Into (F5)”命令会进入调用的子程序，而执行“Step Over (F6)”命令则会直接跳过子程序看成单个源代码行，单步执行将其运行结束。

在继续演示此功能之前，请参照上文所述确保目前只在 blink.c 文件的第 36 行设置了一个断点。

按下 F8 键让程序继续运行然后在断点处停止运行，多次按下“Step Over (F6)”按钮，观察调试器是如何单步执行一行代码的。

如果你改用“Step Into (F5)”，那么调试器将会进入调用的子程序内部。

在上述例子中，调试器进入 gpio_set_level (BLINK_GPIO，0) 代码内部，同时代码窗口快速切换到 gpio.c 驱动文件。

请参阅“next”命令无法跳过子程序的原因 文档以了解 next 命令的潜在局限。

查看并设置内存 要显示或者设置内存的内容，请使用“调试”视图中位于底部的“Memory”选项卡。

在“Memory”选项卡下，我们在内存地址 0x3FF44004 处读取和写入内容。该地址也是 GPIO.OUT_REG 寄存器的地址，可以用来控制（设置或者清除）某个 GPIO 的电平。

关于该寄存器的更多详细信息，请参阅 ESP32 技术参考手册 > IO_MUX 和 GPIO Matrix (GPIO, IO_MUX) [PDF] 章节。
图 47: 使用 “Step Over (F6)” 单步执行代码
图 48: 使用“Step Into (F5)”单步执行代码
同样在 blink.c 项目文件中，两个 gpio_set_level 语句的后面各设置一个断点，单击“Memory”选项卡，然后单击“Add Memory Monitor”按钮，在弹出的对话框中输入 0x3FF44004。

按下 F8 按键恢复程序运行，并观察“Monitor”选项卡。

图 49: 观察内存地址 0x3FF44004 处的某个比特被置高

每按一下 F8，你就会看见内存 0x3FF44004 地址处的一个比特位被翻转 (并且 LED 会改变状态)。

要修改内存的数据，请在“Monitor”选中卡找到待修改的内存地址，如前面观察的结果一样，输入特定比特翻转后的值。当按下回车键后，将即刻看到 LED 的状态发生了改变。

观察和设置程序变量 常见的调试任务是在程序运行期间检查程序中某个变量的值。为了演示这个功能，更新 blink.c 文件，在 blink_task 函数的上面添加一个全局变量的声明，int i，然后在 while(1) 里添加 i++，这样每次 LED 改变状态的时候，变量 i 都会增加 1。

退出调试器，这样就不会与新代码混淆，然后重新构建并烧写代码到 ESP32 中，接着重启调试器，注意，这里不需要我们重启 OpenOCD。

一旦程序停止运行，在代码 i++ 处添加一个断点。

下一步，在“Breakpoints”所在的窗口中，选择“Expressions”选项卡，如果该选项卡不存在，请在顶部菜单栏的 Window > Show View > Expressions 中添加这一选项卡。然后在该选项卡中单击“Add new expression”，并输入 i。

按下 F8 继续运行程序，每次程序停止时，都会看到变量 i 的值在递增。
图 51: 观察程序变量 “i”
如想更改 i 的值，可以在“Value”一栏中输入新的数值。按下 “Resume (F8)” 后，程序将从新输入的数字开始递增 i。

设置条件断点  接下来的内容更为有趣，你可能想在一定条件满足的情况下设置断点，然后让程序停止运行。右击断点打开上下文菜单，选择 “Breakpoint Properties”，将 “Type:” 改选为 “Hardware” 然后在 “Condition:” 一栏中输入条件表达式，例如 i == 2。

图 52: 设置条件断点

如果当前 i 的值小于 2（如果有需要也可以更改这个阈值）并且程序被恢复运行，那么 LED 就会循环闪烁，直到 i == 2 条件成立，最后程序停止在该处。

使用命令行的调试示例  请检查您的目标板是否已经准备好，并加载了 get-started/blink 示例代码，然后按照使用命令行调试 中介绍的步骤配置和启动调试器。最后选择让应用程序在 app_main() 建立的断点处停止运行

```
Temporary breakpoint 1, app_main () at /home/user-name/esp/blink/main/./blink.c:43
43 xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL,
        5, NULL);
```

本小节的示例

1. 浏览代码，查看堆栈和线程
2. 设置和清除断点
3. 暂停和恢复应用程序的运行
4. 单步执行代码
5. 查看并设置内存
第6章 API

6. 观察和设置程序变量
7. 设置条件断点
8. 调试FreeRTOS对象

浏览代码，查看堆栈和线程 当看到（gdb）提示符的时候，应用程序已停止运行，LED也停止闪烁。
要找到代码暂停的位置，输入 l 或者 list 命令，调试器会打印出暂停点（blink.c 代码文件的第 43 行）附近的几行代码

```
(gdb) l
38    }
39    }
40
41    void app_main()
42    {
43        xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL,
44            -5, NULL);
45    }
(gdb)
```

也可以通过输入 l 30, 40 等命令来查看特定行号范围内的代码。
使用 bt 或者 backtrace 来查看哪些函数最终导致了此代码被调用:

```
(gdb) bt
#0  app_main () at /home/user-name/esp/blink/main/.//blink.c:43
#1 0x400d057e in main_task (args=0x0) at /home/user-name/esp/esp-idf/components/
    --esp32/.//cpu_start.c:339
(gdb)
```

输出的第 0 行表示应用程序暂停之前调用的最后一个函数，即我们之前列出的 app_main ()。
app_main () 又被位于 cpu_start.c 文件第 339 行的 main_task 函数调用。
想查看 cpu_start.c 文件中 main_task 函数的上下文，需要输入 frame N，其中 N = 1，因为根据前面的输出，main_task 位于 #1 下:

```
(gdb) frame 1
#1 0x400d057e in main_task (args=0x0) at /home/user-name/esp/esp-idf/components/
    --esp32/.//cpu_start.c:339
339    app_main();
(gdb)
```

输入 l 将显示一段名为 app_main() 的代码 (在第 339 行):

```
(gdb) l
334    }
335   
336   #endif
337   //Enable allocation in region where the startup stacks were located.
338   heap_caps_enable_nonos_stack_heaps();
339   app_main();
340   vTaskDelete(NULL);
341    }
342
(gdb)
```

通过打印前面的一些行，你会看到我们一直在寻找的 main_task 函数:

```
(gdb) l 326, 341
326    static void main_task(void* args)
327    {
328        // Now that the application is about to start, disable boot watchdogs
(gdb)
```

(下页继续)
如果要查看其他代码，可以输入 `i threads` 命令，则会输出目标板上运行的线程列表:

(gdb) i threads
  Id Target Id Frame
  8 Thread 1073411336 (dport) 0x400d0848 in dport_access_init_core (arg=
  "optimized out")
    at /home/user-name/esp/esp-idf/components/esp32./dport_access.c:170
  7 Thread 1073408744 (ipc0) xQueueGenericReceive (xQueue=0x3ffae694,
  pvBuffer=0x0, xTicksToWait=1644638200,
  xJustPeeking=0) at /home/user-name/esp/esp-idf/components/freertos./queue.
  "c:1452
  6 Thread 1073431096 (Tmr Svc) prvTimerTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos./timers.c:445
  5 Thread 1073410208 (ipc1 : Running) 0x4000bfea in ?? ()
  4 Thread 1073432224 (dport) dport_access_init_core (arg=0x0)
    at /home/user-name/esp/esp-idf/components/esp32./dport_access.c:150
  3 Thread 1073431556 (IDLE) prvIdleTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos./tasks.c:3282
  2 Thread 1073412512 (IDLE) prvIdleTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos./tasks.c:3282
* 1 Thread 1073411772 (main : Running) app_main () at /home/user-name/esp/blink/
  main/../blink.c:43
(gdb)

线程列表显示了每个线程最后—个被调用的函数以及所在的 C 源文件名（如果存在的话）。
您可以通过输入 `thread N` 进入特定的线程。其中 N 是线程 ID。我们进入 5 号线程来看一下它是如何工作的:

(gdb) thread 5
[Switching to thread 5 (Thread 1073410208)]
#0 0x4000bfea in ?? ()
(gdb)

然后查看回溯:

(gdb) bt
#0 0x4000bfea in ?? ()
#1 0x40004a85 in vPortCPUReleaseMutex (mux=<optimized out>) at /home/user-name/
  "esp/esp-idf/components/freertos./port.c:415
#2 0x40004f3c8 in vTaskSwitchContext () at /home/user-name/esp/esp-idf/components/
  "freertos./tasks.c:2846
#3 0x4000532b in _frxt_dispatch ()
#4 0x4000393c in xPortStartScheduler () at /home/user-name/esp/esp-idf/components/
  "freertos./port.c:222

(下页继续)
如上所示，回溯可能会包含多个条目，方便查看直至目标停止运行的函数调用顺序。如果找不到某个函数的源码文件，将会使用问号 ?? 替代，这表示该函数是以二进制格式提供的。像 0x4000bfea 这样的值是被调用函数所在的内存地址。

使用诸如 bt, i threads, thread N 和 list 命令可以浏览整个应用程序的代码。这给单步调试代码和设置断点带来很大的便利，下面将一一展开来讨论。

### 设置和清除断点
在调试时，我们希望能够在关键的代码行停止应用程序，然后检查特定的变量，内存，寄存器和外设的状态。为此我们需要使用断点，以便在同一行代码处快速访问和停止应用程序。

我们在控制 LED 状态发生变化的两处代码行分别设置一个断点。基于以上代码列表，这两处分别为第 33 和 36 代码行。使用命令 break M 设置断点，其中 M 是具体的代码行:

```plaintext
(gdb) break 33
Breakpoint 2 at 0x400db6f6: file /home/user-name/esp/blink/main./blink.c, line 33.
(gdb) break 36
Breakpoint 3 at 0x400db704: file /home/user-name/esp/blink/main./blink.c, line 36.
```

输入命令 c，处理器将运行并在断点处停止。再次输入 c 将使其再次运行，并在第二个断点处停止。依此类推:

```plaintext
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB6F6 (active)  APP_CPU: PC=0x400D10D8
Breakpoint 2, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main./blink.c:33
   33 gpio_set_level(BLINK_GPIO, 0);
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB704 (active)  APP_CPU: PC=0x400D10D8
Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main./blink.c:36
   36 gpio_set_level(BLINK_GPIO, 1);
(gdb)
```

只有在输入命令 c 恢复程序运行后才能看到 LED 改变状态。

查看已设置断点的数量和位置，请使用命令 info break:

```plaintext
(gdb) info break
Num Type Disp Enb Address What
2 breakpoint keep y 0x400db6f6 in blink_task at /home/user-name/esp/blink/main./blink.c:33
   breakpoint already hit 1 time
3 breakpoint keep y 0x400db704 in blink_task at /home/user-name/esp/blink/main./blink.c:36
   breakpoint already hit 1 time
(gdb)
```

请注意，断点序号（在 Num 栏列出）从 2 开始，这是因为调试器启动时执行 thb app_main 命令已经在 app_main() 函数处建立了第一个断点。由于它是一个临时断点，已经被自动删除，所以没有被列出。
要删除一个断点，请输入 delete N 命令（或者简写成 d N），其中 N 代表断点序号:

```plaintext
(gdb) delete 1
No breakpoint number 1.
(gdb) delete 2
(gdb)
```

更多关于断点的信息，请参阅可用的断点和观察点 和关于断点的补充知识。

**暂停和恢复应用程序的运行**  在调试时，可以恢复程序运行并输入代码等待某个事件发生或者保持无限循环而不设置任何断点。对于后者，想要返回调试模式，可以通过输入 Ctrl+C 手动中断程序的运行。

在之前，请删除所有的断点，然后输入 c 恢复程序运行。接着输入 Ctrl+C，应用程序会停止在某个随机的位置，此时 LED 也将停止闪烁。调试器会打印如下信息:

```plaintext
(gdb) c
Continuing.
^C
```

在上图所示的情况下，应用程序已经在 freertos_hooks.c 文件的第 52 行暂停运行，现在您可以通过输入 c 再次将其恢复运行或者进行如下所述的一些调试工作。

**单步执行代码** 我们还可以使用 step 和 next 命令（可以简写成 s 和 n）单步执行代码，这两者之间的区别是执行 “step” 命令进入调用的子程序内部，而执行 “next” 命令则会直接将子程序看成单个源码行，单步就能将其运行结束。

在继续演示此功能之前，请使用前面介绍的 break 和 delete 命令，确保目前只在 blink.c 文件的第 36 行设置了一个断点:

```plaintext
(gdb) info break
Num Type Disp Enb Address What
 3 breakpoint keep y 0x400db704 in blink_task at /home/user-name/esp/blink/main/./blink.c:36
breakpoint already hit 1 time
(gdb)
```

输入 c 恢复程序运行然后等它在断点处停止运行:

```plaintext
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB754 (active) APP_CPU: PC=0x400D1128
Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:36
36 gpio_set_level(BLINK_GPIO, 1);
(gdb)
```

然后输入 n 多次，观察调试器是如何单步执行一行代码的:

```plaintext
(gdb) n
Target halted. PRO_CPU: PC=0x400DB756 (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB758 (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400D04C (active) APP_CPU: PC=0x400D1128
```
如果你输入 \texttt{s}，那么调试器将进入子程序:

```
(gdb) s
```

```
Target halted. PRO_CPU: PC=0x400DB748 (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB74B (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04C (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04F (active) APP_CPU: PC=0x400D1128
gpio_set_level (gpio_num=GPIO_NUM_4, level=0) at /home/user-name/esp/esp-idf/\rightarrow components/driver/./gpio.c:183
183 GPIO_CHECK(GPIO_IS_VALID_OUTPUT_GPIO(gpio_num), "GPIO output gpio_num error", ESP_ERR_INVALID_ARG);
(gdb)
```

上述例子中，调试器进入 \texttt{gpio_set_level} (BLINK_GPIO, 0) 代码内部，同时代码窗口快速切换到 gpio.c 驱动文件。

请参阅 “next” 命令无法跳过子程序的原因 文档以了解 next 命令的潜在局限。

**查看并设置内存** 使用命令 \texttt{x} 可以显示内存的内容，配合其余参数还可以调整所显示内存位置的格式和数量。运行 \texttt{help x} 可以查看更多相关细节。与 \texttt{x} 命令配合使用的命令是 \texttt{set}，它允许你将值写入内存。

为了演示 \texttt{x} 和 \texttt{set} 的使用，我们将在内存地址 0x3FF44004 处读取和写入内容。该地址也是 GPIO_OUT_REG 寄存器的地址，可以用来控制（设置或者清除）某个 GPIO 的电平。

关于该寄存器的更多详细信息，请参阅 ESP32 技术参考手册 > IO_MUX 和 GPIO Matrix (GPIO, IO_MUX) [PDF] 章节。

同样在 blink.c 项目文件中，在两个 \texttt{gpio_set_level} 语句的后面各设置一个断点。输入两次 \texttt{c} 命令后停止在断点处，然后输入 \texttt{x} /1wx 0x3FF44004 来显示 GPIO_OUT_REG 寄存器的值:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB75B (active) APP_CPU: PC=0x400D1128
Breakpoint 2, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./\rightarrow blink.c:34
34 vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000000
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB75B (active) APP_CPU: PC=0x400D1128
Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./\rightarrow blink.c:37
37 vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000010
(gdb)
```
如果闪烁的 LED 连接到了 GPIO4，那么每次 LED 改变状态时你会看到第 4 比特被翻转:

<table>
<thead>
<tr>
<th>0x3ff44004: 0x00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>0x3ff44004: 0x00000010</td>
</tr>
</tbody>
</table>

现在，当 LED 熄灭时，与之对应地会显示 0x3ff44004: 0x00000000，尝试使用 set 命令向相同的内存地址写入 0x00000010 来将该比特置高:

```
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000000
```

在输入 set (unsigned int)0x3FF44004=0x00000010 命令后，你会立即看到 LED 亮起。

**观察和设置程序变量** 常见的调试任务是在程序运行期间检查程序中某个变量的值，为了能够演示这个功能，更新 blink.c 文件，在 blink_task 函数的上面添加一个全局变量的声明 int i，然后在 while(1) 里添加 i++，这样每次 LED 改变状态的时候，变量 i 都会增加 1。退出调试器，这样就不会与新代码混淆，然后重新构建并烧写代码到 ESP32 中，接着重启调试器。注意，这里不需要我们重启 OpenOCD。

一旦程序停止运行，输入命令 watch i:

```
(gdb) watch i
Hardware watchpoint 2: i
```

这会把所有变量 i 发生改变的代码处插入所谓的“观察点”。现在输入 continue 命令来恢复应用程序的运行并观察它停止:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active)  APP_CPU: PC=0x400D0811
[New Thread 1073432196]
Program received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 1073432196]
0x400db751 in blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
   → blink.c:33
33   i++;
```

多次恢复程序运行后，变量 i 的值会增加，现在你可以输入 print i (简写 p i) 来查看当前 i 的值:

```
(gdb) p i
$1 = 3
```

要修改 i 的值，请使用 set 命令，如下所示（可以将其打印输出来查看是否已修改）:

```
(gdb) set var i = 0
(gdb) p i
$3 = 0
```

最多可以使用两个观察点，详细信息请参阅可用的断点和观察点。

**设置条件断点** 接下来的内容更为有趣，你可能想在一定条件满足的情况下设置断点。请先删除已有的断点，然后尝试如下命令:

```
(gdb) continue
(gdb) break i = 3
(gdb) continue
```

```
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active)  APP_CPU: PC=0x400D0811
[New Thread 1073432196]
Program received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 1073432196]
0x400db751 in blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
   → blink.c:33
33   i++;
```

当前 i 的值是 3，现在你可以输入 print i 来查看当前 i 的值:

```
(gdb) p i
$1 = 3
```

要修改 i 的值，请使用 set 命令，如下所示（可以将其打印输出来查看是否已修改）:

```
(gdb) set var i = 0
(gdb) p i
$3 = 0
```
(gdb) break blink.c:34 if (i == 2)
Breakpoint 3 at 0x400db753: file /home/user-name/esp/blink/main././blink.c, line 34.
(gdb)

以上命令在 blink.c 文件的 34 处设置了一个条件断点，当 i == 2 条件满足时，程序会停止运行。
如果当前 i 的值小于 2 并且程序被恢复运行，那么 LED 就会循环闪烁，直到 i == 2 条件成立，最后程序停止在该处:

(gdb) set var i = 0
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB755 (active) APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB755 (active) APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB755 (active) APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB755 (active) APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB755 (active) APP_CPU: PC=0x400D112C

(gdb) break blink.c:34 if (i == 2)
Breakpoint 3 at 0x400db753: file /home/user-name/esp/blink/main././blink.c, line 34.
(gdb)

调试 FreeRTOS 对象：该部分内容或许可以帮助您调试 FreeRTOS 任务交互。需要调试 FreeRTOS 任务交互的用户可使用 GDB 命令 freertos。该命令非 GDB 原生命令，而是来自于 Python 扩展模块 freertos-gdb，其包含一系列子命令:

(gdb) freertos
"freertos" 后面必须紧跟子命令的名称
freertos 子命令如下:
freertos queue -- 打印当前队列信息
freertos semaphore -- 打印当前信号量信息
freertos task -- 打印当前任务及其状态
freertos timer -- 打印当前定时器信息

点击 https://pypi.org/project/freertos-gdb 链接了解此扩展模块的详细信息。

备注: ESP-IDF 在安装 Python 包时会自动安装 freertos-gdb Python 模块，详情请参考第三步: 设置工具。
如果使用 idf.py gdb 命令运行 GDB，FreeRTOS 扩展会自动加载。也可以使用 GDB 内部命令 python import freertos_gdb 使能该模块。
请保证使用 Python 3.6 及以上版本，该版本具有 Python 共享库。

命令的帮助信息：目前所介绍的都是些非常基础的命令，目的在于让您快速上手 JTAG 调试。如果想获得特定命令的语法和功能相关的信息，请在 (gdb) 提示符下输入 help 和命令名:

(gdb) help next
Step program, proceeding through subroutine calls.
Usage: next [N]
Unlike "step", if the current source line calls a subroutine, this command does not enter the subroutine, but instead steps over the call, in effect treating it as a single source line.
(gdb)

只需输入 help 命令，即可获得高级命令列表，帮助你了解更多详细信息。此外，还可以参考一些 GDB 命令速查表，比如 https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf。虽然不是所有命令都适用于嵌入式环境，但还是会有所帮助。
结束调试会话 输入命令 `q` 可以退出调试器:

```
(gdb) q
A debugging session is active.

Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) y
Detaching from program: /home/user-name/esp/blink/build/blink.elf, Remote target
Ending remote debugging.
user-name@computer-name:~/esp/blink$
```

- 使用调试器
- 调试示例
- 注意事项和补充内容
- 应用层跟踪库
- ESP-Prog 调试板介绍

### 4.18 链接器脚本生成机制

#### 4.18.1 概述

ESP32 中有多个用于存放代码和数据的内存区域。代码和只读数据默认存放在 flash 中，可写数据存放在 RAM 中。不过有时，用户必须更改默认存放区域。

例如:

- 将关键代码存放到 RAM 中以提高性能;
- 将可执行代码存放到 IRAM 中，以便在缓存被禁用时运行这些代码;
- 将代码存放到 RTC 存储器中，以便在 wake stub 中使用;
- 将代码存放到 RTC 内存中，以便 ULP 协处理器使用。

链接器脚本生成机制可以让用户指定代码和数据在 ESP-IDF 组件中的存放区域。组件包含如何存放符号、目标或完整库的信息。在构建应用程序时，组件中的这些信息会被收集、解析并处理，生成的存放规则用于链接应用程序。

#### 4.18.2 快速上手

本段将指导如何使用 ESP-IDF 的即用方案，快速将代码和数据放入 RAM 和 RTC 存储器中。

假设用户有:

```
components
└── my_component
    ├── CMakeLists.txt
    ├── Kconfig
    └── src/
        ├── my_src1.c
        │ └── my_src1.o
        ├── my_src2.c
        │ └── my_src2.o
        └── my_src3.c
            └── my_src3.o
```

- 名为 `my_component` 的组件，在构建过程中存储为 `libmy_component.a` 库文件
- 库文件包含的三个源文件: `my_src1.c`、`my_src2.c` 和 `my_src3.c`。编译后分别为 `my_src1.o`、`my_src2.o` 和 `my_src3.o`
• 在 my_src1.o 中定义 my_function1 功能；在 my_src2.o 中定义 my_function2 功能
• 在 my_component 下 Kconfig 中存在布尔类型配置 PERFORMANCE_MODE (y/n) 和整数类型配置 PERFORMANCE_LEVEL（范围是 0-3）

创建和指定链接器片段文件

首先，用户需要创建链接器片段文件。链接器片段文件是一个扩展名为 .lf 的文本文件，想要存放的位置信息会写入该文件内。文件创建成功后，需要将其呈现构建系统中。ESP-IDF 支持的构建系统指南如下：

在组件目录的 CMakeLists.txt 文件中，指定 idf_component_register 调用引数 LDFRAGMENTS 的值。LDFRAGMENTS 可以为绝对路径，也可为组件目录的相对路径，指向已创建的链接器片段文件。

```cpp
# 相对于组件的 CMakeLists.txt 的文件路径
idf_component_register([...)
  LDFRAGMENTS "path/to/linker_fragment_file.lf" "path/to/
               another_linker_fragment_file.lf"
  ...
]
```

指定存放区域

可以按照下列粒度指定存放区域：
• 目标文件 (.obj 或 .o 文件)
• 符号（函数/变量）
• 库 (.a 文件)

存放目标文件 假设整个 my_src1.o 目标文件对性能至关重要，所以最好把该文件放在 RAM 中。另外，my_src2.o 目标文件包含从睡眠唤醒所需的符号，因此需要将其存放到 RTC 存储器中。

在链接器片段文件中可以写入以下内容：

```cpp
[mapping:my_component]
archive: libmy_component.a
entries:
  my_src1 (noflash)  # 将所有 my_src1 代码和只读数据存放在 IRAM 和 DRAM 中
  my_src2 (rtc)     # 将所有 my_src2 代码、数据和只读数据存放在 RTC 快速 RAM 中
```

那么 my_src3.o 放在哪里呢？由于未指定存放区域，my_src3.o 会存放到默认区域。更多关于默认存放区域的信息，请查看这里。

存放符号 继续上文的例子，假设 object1.o 目标文件定义的功能中，只有 my_function1 影响到性能：object2.o 目标文件中只有 my_function2 需要在芯片从深度睡眠中唤醒后运行。要实现该目的，可在链接器片段文件中写入以下内容：

```cpp
[mapping:my_component]
archive: libmy_component.a
entries:
  my_src1:my_function1 (noflash)
  my_src2:my_function2 (rtc)
```

my_src1.o 和 my_src2.o 中的其他函数以及整个 object3.o 目标文件会存放到默认区域。要指定数据的存放区域，仅需将上文的函数名替换为变量名即可，如:

```cpp
my_src1:my_variable (noflash)
```
**存放整个库** 在这个例子中，假设整个组件库需存放到 RAM 中，可以写入以下内容存放整个库：

```
[mapping:my_component]
archive: libmy_component.a
entries:
  * (noflash)
```

类似的，写入以下内容可以将整个组件存放到 RTC 存储器中：

```
[mapping:my_component]
archive: libmy_component.a
entries:
  * (rtc)
```

**根据具体配置存放** 假设只有在某个条件为真时，比如 `CONFIG_PERFORMANCE_MODE == y` 时，整个组件库才有特定存放区域，可以写入以下内容实现：

```
[mapping:my_component]
archive: libmy_component.a
entries:
  if PERFORMANCE_MODE == y:
    * (noflash)
  else:
    * (default)
```

来看一种更复杂的情况。假设 “`CONFIG_PERFORMANCE_LEVEL == 1“ 时，只有 object1.o 存放到 RAM 中；`CONFIG_PERFORMANCE_LEVEL == 2` 时，object1.o 和 object2.o 会存放到 RAM 中；`CONFIG_PERFORMANCE_LEVEL == 3` 时，库中的所有目标文件都会存放到 RAM 中，以上三个条件为假时，整个库会存放到 RTC 存储器中。虽然这种使用场景很罕见，不过，还是可以通过以下方式实现：

```
[mapping:my_component]
archive: libmy_component.a
entries:
  if PERFORMANCE_LEVEL == 1:
    my_src1 (noflash)
  elif PERFORMANCE_LEVEL == 2:
    my_src1 (noflash)
    my_src2 (noflash)
  elif PERFORMANCE_LEVEL == 3:
    my_src1 (noflash)
    my_src2 (noflash)
    my_src3 (noflash)
  else:
    * (rtc)
```

也可以嵌套条件检查。以下内容与上述片段等效：

```
[mapping:my_component]
archive: libmy_component.a
entries:
  if PERFORMANCE_LEVEL <= 3 && PERFORMANCE_LEVEL > 0:
    if PERFORMANCE_LEVEL >= 1:
      object1 (noflash)
    if PERFORMANCE_LEVEL >= 2:
      object2 (noflash)
```

(下页继续)
默认存放区域

到目前为止，“默认存放区域”在未指定 rtc 和 noflash 存放规则时才会作为备选方案使用。需要注意的是，noflash 或者 rtc 标记不仅仅是关键字，实际上还是被称作片段的实体，确切地说是协议。

与 rtc 和 noflash 类似，还有一个默认协议，定义了默认存放规则。顾名思义，该协议规定了代码和数据通过存放的区域，即代码和变量存放在 flash 中，变量存放在 RAM 中。更多关于默认协议的信息，请见这里。

备注：使用链接器脚本生成机制的 IDF 组件示例，请参阅 freertos/CMakeLists.txt。为了提高性能，freertos 使用链接器脚本生成机制，将其目标文件存放到 RAM 中。

4.18.3 链接器脚本生成机制内核

链接是将 C/C++ 源文件转换或可执行文件的最后一步。链接由工具链的链接器完成，接受指定代码和数据存放区域等信息的链接脚本。链接器脚本生成机制的转换过程类似，区别在于传输给链接器的链接脚本根据 (1) 收集的链接器片段文件 和 (2) 链接器脚本模板 动态生成。

备注：执行链接器脚本生成机制的工具存放在 tools/ldgen 之下。

链接器片段文件

如快速入门指南所述，片段文件是扩展名为 .lf 的简单文本文件，内含存放可存放区域的信息。不过，这是对片段文件所包含内容的简化版描述。实际上，片段文件内包含的是“片段”。片段是实体，包含多条信息，这些信息放在一起组成了存放规则，说明目标文件各个段在二进制输出文件中的存放位置。片段一共有三种，分别是 段、协议 和 映射。

语法 三种片段类型使用同一种语法：

```
[type:name]
key: value
key:
  value
  value
  value
...
```

- 类型：片段类型，可以为 段、协议 或 映射。
- 名称：片段名称，指定片段类型的片段名称应唯一。
- 键值：片段内容。每个片段类型可支持不同的键值和不同的键值语法。
  - 在 段 和 协议 中，仅支持 entries 键。
  - 在 映射 中，支持 archive 和 entries 键。

备注：多个片段的类型和名称相同时会引发异常。
条件检查

条件检查使得链接器脚本生成机制可以感知配置。含有配置值的表达式是否为真，决定了使用哪些特定键值。检查使用的是 kconfiglib 脚本的 eval_string，遵循该脚本要求的语法和局限性，支持：

- **比较**
  - 小于 <
  - 小于等于 <=
  - 大于 >
  - 大于等于 >=
  - 等于 =
  - 不等于 !=

- **逻辑**
  - 或 ||
  - 和 &&
  - 取反 ~

- **分组**
  - 圆括号 ()

条件检查和其他语言中的 if...elseif/elif...else 块作用一样。键值和完整片段都可以进行条件检查。以下两个示例效果相同：

```python
# 配置决定键值
[type:name]
key_1:
  if CONDITION = y:
    value_1
  else:
    value_2
key_2:
  if CONDITION = y:
    value_a
  else:
    value_b

# 完整片段的定义决定键值
if CONDITION = y:
  [type:name]
  key_1:
    value_1
  key_2:
    value_a
else:
  [type:name]
  key_1:
    value_2
  key_2:
    value_b
```

**注释**

链接器片段文件中的注释以 # 开头。和在其他语言中一样，注释提供了有用的描述和资料，在处理过程中会被忽略。

**类型段**

段定义了 GCC 编译器输出的一系列目标文件段，可以是默认段（如 .text, .data），也可以是用户通过 __attribute__ 关键字定义的段。
Chapter 4. API 指南

`*` 表示段列表开始，且当前段为列表中的第一个段，这种表达方式更加推荐。

```plaintext
[sections:name]
entries:
  .section+
  .section
  ...
```

示例：

```plaintext
# 不推荐的方式
[sections:text]
entries:
  .text
  .text.*
  .literal
  .literal.*

# 推荐的方式，效果与上面等同
[sections:text]
entries:
  .text+     # 即 .text 和 .text.*
  .literal+  # 即 .literal 和 .literal.*
```

协议

协议定义了每个段对应的 目标。

```plaintext
[scheme:name]
entries:
  sections -> target
  sections -> target
  ...
```

示例：

```plaintext
[scheme:noflash]
entries:
  text -> iram0_text       # text 段下的所有条目均归入 iram0_text
  rodata -> dram0_data     # rodata 段下的所有条目均归入 dram0_data
```

默认协议

注意，有一个默认的协议很特殊，特殊在于包罗存放规则都是根据这个协议中的条目生成的。这意味着，如果该协议有一条条目是 text -> flash_text，则将为目标 flash_text 生成如下的存放规则：

```plaintext
* (.literal .literal.* .text .text.*)
```

这些生成的包罗规则将用于未指定映射规则的情况。

默认协议在 esp_system/app.lf 文件中定义。快速上手指南中提到的内置noflash 协议和 rtc 协议也在该文件中定义。

映射

映射定义了可映射实体（即目标文件、函数名、变量名和库）对应的协议。

```plaintext
[mapping]
archive: archive                      # 构建后输出的库文件名称（即 libxxx.a)
entries:
  object:symbol (scheme)               # 符号
  object (scheme)                      # 目标
  * (scheme)                           # 库
```

有三种存放粒度：
Chapter 4. API 指南

- 符号：指定了目标文件名称和符号名称。符号名称可以是函数名或变量名。
- 目标：只指定目标文件名称。
- 库：指定 *，即某个库下面所有目标文件的简写表达法。

为了更好地理解条目的含义，请看一个按目标存放的例子。

```object (scheme)
```

根据条目定义，将这个协议展开：

```object (sections -> target,
sections -> target,
...)
```

再根据条目定义，将这个段展开：

```object (.section,
.section,
... -> target, # 根据目标文件将这里所列出的所有段放在该目标位置
.section,
.section,
... -> target, # 同样的方法指定其他段
...)
```
# 直至所有段均已展开

示例：

```[mapping:map]
archive: libfreertos.a
entries:
  * (noflash)
```

除了实体和协议，条目中也支持指定如下符号：（注：<> = 参数名称，[] = 可选参数）

1. **ALIGN(<alignment>[, pre, post])**
   根据 alignment 中指定的数字对齐存放区域。根据是否指定 pre 和 post，或两者都指
   定，在输入段描述（生成于映射条目）的前面和/或后面生成。

2. **SORT([<sort_by_first>, <sort_by_second>])**
   在输入段描述中输出 SORT_BY_NAME, SORT_BY_ALIGNMENT, SORT_BY_INIT_PRIORITY 或 SORT.
   sort_by_first 和 sort_by_second 的值可以是 name, alignment, init_priority。
   如果既没指定 sort_by_first 也没指定 sort_by_second，则输入段会按照名称
   排序，如果两者都指定了，那么按套排序会遵循 https://sourceware.org/binutils/docs/ld/
   Input-Section-Wildcards.html 中的规则。

3. **KEEP()**
   用 KEEP 命令包围输入段描述，从而防止链接器丢弃存放区域。更多细节请参考 https:
   //sourceware.org/binutils/docs/ld/Input-Section-Keep.html

4. **SURROUND(<name>)**

在存放区域的前面和后面生成符号，生成的符号遵循_<name>_start 和_<name>_end 的
命名方式，例如，如果 name == sym1

在添加标志时，协议中需要指定具体的 section -> target。对于多个 section -> target，使
用逗号作为分隔符，例如：

```# 注意
# A. entity-scheme 后使用分号
# B. section2 -> target2 前使用逗号
# C. 在 scheme1 条目中定义 section1 -> target1 和 section2 -> target2
entity1 (scheme1);
```
合并后，如下的映射:

```
[mapping:name]
archive: lib1.a
entries:
  obj1 (noflash);
  rodata -> dram0_data KEEP() SORT() ALIGN(8) SURROUND(my_sym)
```

会在链接器脚本上生成如下输出:

```
._ = ALIGN(8)
__my_sym_start = ABSOLUTE(.)
KEEP(lib1.a:obj1.*( SORT(.rodata) SORT(.rodata.* ) ))
__my_sym_end = ABSOLUTE(.)
```

注意，正如在 `flag` 描述中提到的，ALIGN 和 SURROUND 的使用对顺序敏感，因此如果将两者顺序调换后用到相同的映射片段，则会生成:

```
__my_sym_start = ABSOLUTE(.)
._ = ALIGN(8)
KEEP(lib1.a:obj1.*( SORT(.rodata) SORT(.rodata.* ) ))
__my_sym_end = ABSOLUTE(.)
```

**按符号存放** 按符号存放可通过编译器标志 `-ffunction-sections` 和 `-ffdata-sections` 实现。ESP-IDF 默认用这些标志编译，用户若选择移除标志，便不能按符号存放。另外，即便有标志，也会其他限制，具体取决于编译器输出的段。

比如，使用 `-ffunction-sections`，针对每个函数会输出单独的段。段的名称可以预测，即 `.text.{func_name}` 和 `.data.{func_name}`。但是功能内的字符串并非如此，因为字符串会进入字符串池，或者使用生成的段名称。

使用 `-ffdata-sections`，对全局数据来说编译器可输出 `.data.{var_name}`、`.rodata.{var_name}` 或 `.bss.{var_name}`；因此类型 I 映射词条可以适用。但是，功能中声明的静态数据并非如此，生成的段名称是将变量名称和其他信息混合。

**链接器脚本模板**

链接器脚本模板是指定存放规则的存放位置的框架，与其他链接器脚本没有本质区别，但带有特定的标记语法，可以指示存放生成的存放规则的位置。

如需引用一个目标标记下的所有存放规则，请使用以下语法:

```
mapping[target]
```

示例:

以下示例是某个链接器脚本模板的摘录，定义了输出段 `.iram0.text`，该输出段包含一个引用目标 `.iram0_text` 的标记。

```
.iram0.text :
{
  /* 标记 IRAM 空间不足 */
  __iram_text_start = ABSOLUTE(.);

  /* 引用 iram0_text */
  mapping[iram0_text]
```

(下页继续)
假设链接器脚本生成器收集到了以下片段定义:

```c
[sections:text]
 .text+
 .literal+
[sections:iram]
 .iram1+
[scheme:default]
 entries:
   text -> flash_text
   iram -> iram0_text
[scheme:noflash]
 entries:
   text -> iram0_text

[mapping:freertos]
 archive: libfreertos.a
 entries:
   * (noflash)

然后生成的链接器脚本的相应摘录如下:

```c
.iram0.text :
{
   /* 标记 IRAM 空间不足 */
   _iram_text_start = ABSOLUTE(.);

   /* 处理片段生成的存放规则，存放于模板标记的位置处 */
   "libfreertos.a:(.literal .text .literal.* .text.*)"
   _iram_text_end = ABSOLUTE(.);
} > iram0_0_seg
```

*libfreertos.a:(.literal .text .literal.* .text.*)

这是根据 freertos 映射的 * (noflash) 条目生成的规则。libfreertos.a 库下所有目标文件的所有 text 段会收集到 iram0_text 目标下 (按照 noflash 协议), 并放在模板中被 iram0_text 标记的地方。

* (.iram1 .iram1.*)

这是根据默认协议条目 iram -> iram0_text 生成的规则。默认协议指定了 iram -> iram0_text 条目。因此生成的规则同样也放在被 iram0_text 标记的地方。由于该规则是根据默认协议生成的，因此在同一目标下收集的所有规则下排在第一位。

目前使用的链接器脚本模板是 esp_system/ld/esp32/sections.ld.in，生成的脚本存放在构建目录下。

**将链接器脚本片段文件语法迁移至 ESP-IDF v5.0 适应版本**

ESP-IDF v5.0 中将不再支持 ESP-IDF v3.x 中链接器脚本片段文件的旧式语法。在迁移的过程中需注意以下几点:
4.19 lwIP

ESP-IDF uses the open source lwIP lightweight TCP/IP stack. The ESP-IDF version of lwIP (esp-lwip) has some modifications and additions compared to the upstream project.

4.19.1 Supported APIs

ESP-IDF supports the following lwIP TCP/IP stack functions:

- BSD Sockets API
- Netconn API is enabled but not officially supported for ESP-IDF applications

Adapted APIs

Some common lwIP “app” APIs are supported indirectly by ESP-IDF:

- DHCP Server & Client are supported indirectly via the ESP-NETIF functionality
- Simple Network Time Protocol (SNTP) is supported via the lwip/include/apps/sntp/sntp.h lwip/lwip/src/include/lwip/apps/sntp.h functions (see also SNTP 時間同步)
- ICMP Ping is supported using a variation on the lwIP ping API. See ICMP Echo.
- NetBIOS lookup is available using the standard lwIP API, protocols/http_server/restful_server has an option to demonstrate using NetBIOS to look up a host on the LAN.
- mDNS uses a different implementation to the lwIP default mDNS (see mDNS 服務), but lwIP can look up mDNS hosts using standard APIs such as gethostbyname() and the convention hostname.local, provided the CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES setting is enabled.

4.19.2 BSD Sockets API

The BSD Sockets API is a common cross-platform TCP/IP sockets API that originated in the Berkeley Standard Distribution of UNIX but is now standardized in a section of the POSIX specification. BSD Sockets are sometimes called POSIX Sockets or Berkeley Sockets.

As implemented in ESP-IDF, lwIP supports all of the common usages of the BSD Sockets API.

References

A wide range of BSD Sockets reference material is available, including:

- Single UNIX Specification BSD Sockets page
- Berkeley Sockets Wikipedia page

Examples

A number of ESP-IDF examples show how to use the BSD Sockets APIs:

- protocols/sockets/tcp_server
- protocols/sockets/tcp_client
- protocols/sockets/udp_server
- protocols/sockets/udp_client
- protocols/sockets/udp_multicast
• **protocols/http_request** (Note: this is a simplified example of using a TCP socket to send an HTTP request. The *ESP HTTP Client* is a much better option for sending HTTP requests.)

**Supported functions**

The following BSD socket API functions are supported. For full details see `lwip/lwip/src/include/lwip/sockets.h`.

- `socket()`  
- `bind()`  
- `accept()`  
- `shutdown()`  
- `getpeername()`  
- `getsockopt()` & `setsockopt()` (see *Socket Options*)  
- `close()` (via 虚拟文件系统组件)  
- `read()`, `readv()`, `write()`, `writev()` (via 虚拟文件系统组件)  
- `recv()`, `recvmsg()`, `recvfrom()`  
- `send()`, `sendmsg()`, `sendto()`  
- `select()` (via 虚拟文件系统组件)  
- `poll()` (Note: on ESP-IDF, `poll()` is implemented by calling `select` internally, so using `select()` directly is recommended if a choice of methods is available.)  
- `fcntl()` (see *fcntl*)

Non-standard functions:

- `ioctl()` (see *ioctl*)

`备注`: Some lwIP application sample code uses prefixed versions of BSD APIs, for example `lwip_socket()` instead of the standard `socket()`. Both forms can be used with ESP-IDF, but using standard names is recommended.

**Socket Error Handling**

BSD Socket error handling code is very important for robust socket applications. Normally the socket error handling involves the following aspects:

- Detecting the error.  
- Getting the error reason code.  
- Handle the error according to the reason code.

In lwIP, we have two different scenarios of handling socket errors:

- Socket API returns an error. For more information, see *Socket API Errors*.  
- `select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)` has exception descriptor indicating that the socket has an error. For more information, see *select() Errors*.

**Socket API Errors**

**The error detection**

- We can know that the socket API fails according to its return value.

**Get the error reason code**

- When socket API fails, the return value doesn’ t contain the failure reason and the application can get the error reason code by accessing `errno`. Different values indicate different meanings. For more information, see `<Socket Error Reason Code>`.

Example:
int err;
int sockfd;

if (sockfd = socket(AF_INET, SOCK_STREAM, 0) < 0) {
    // the error code is obtained from errno
    err = errno;
    return err;
}

select() Errors

The error detection

- Socket error when select() has exception descriptor

Get the error reason code

- If the select indicates that the socket fails, we can’t get the error reason code by accessing errno, instead we should call getsockopt() to get the failure reason code. Because select() has exception descriptor, the error code will not be given to errno.

备注: getsockopt function prototype int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen). Its function is to get the current value of the option of any type, any state socket, and store the result in optval. For example, when you get the error code on a socket, you can get it by getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen).

Example:

```c
int err;
if (select(sockfd + 1, NULL, NULL, &exfds, &tval) <= 0) {
    err = errno;
    return err;
} else {
    if (FD_ISSET(sockfd, &exfds)) {
        // select() exception set using getsockopt()
        int optlen = sizeof(int);
        getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen);
        return err;
    }
}
```

Socket Error Reason Code Below is a list of common error codes. For more detailed list of standard POSIX/C error codes, please see newlib/errno.h <https://github.com/espressif/newlib-esp32/blob/master/newlib/libc/include/sys/errno.h> and the platform-specific extensions newlib/platform_include/errno.h
## Error codes

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECONNREFUSED</td>
<td>Connection refused</td>
</tr>
<tr>
<td>EADDRINUSE</td>
<td>Address already in use</td>
</tr>
<tr>
<td>ECONNABORTED</td>
<td>Software caused connection abort</td>
</tr>
<tr>
<td>ENETUNREACH</td>
<td>Network is unreachable</td>
</tr>
<tr>
<td>ENETDOWN</td>
<td>Network interface is not configured</td>
</tr>
<tr>
<td>ETimedout</td>
<td>Connection timed out</td>
</tr>
<tr>
<td>EHOSTDOWN</td>
<td>Host is down</td>
</tr>
<tr>
<td>EHOSTUNREACH</td>
<td>Host is unreachable</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Connection already in progress</td>
</tr>
<tr>
<td>EALREADY</td>
<td>Socket already connected</td>
</tr>
<tr>
<td>EDESTADDRREQ</td>
<td>Destination address required</td>
</tr>
<tr>
<td>EPROTONOSUPPORT</td>
<td>Unknown protocol</td>
</tr>
</tbody>
</table>

### Socket Options

The `getsockopt()` and `setsockopt()` functions allow getting/setting per-socket options.

Not all standard socket options are supported by lwIP in ESP-IDF. The following socket options are supported:

#### Common options

Used with level argument `SOL_SOCKET`.

- `SO_REUSEADDR` (available if `CONFIG_LWIP_SO_REUSE` is set, behavior can be customized by setting `CONFIG_LWIP_SO_REUSE_RTOALL`)
- `SO_KEEPALIVE`
- `SO_BROADCAST`
- `SO_ACCEPTCONN`
- `SO_RCVBUF` (available if `CONFIG_LWIP_SO_RCVBUF` is set)
- `SO_SNDBUF`, `SO_RCVTIMEO`
- `SO_ERROR` (this option is only used with `select()`, see `Socket Error Handling`)
- `SO_TYPE`
- `SO_NO_CHECK` (for UDP sockets only)

#### IP options

Used with level argument `IPPROTO_IP`.

- `IP_TOS`
- `IP_TTL`
- `IP_PKTINFO` (available if `CONFIG_LWIP_NETBUF_RECVINFO` is set)

For multicast UDP sockets:

- `IP_MULTICAST_IF`
- `IP_MULTICAST_LOOP`
- `IP_MULTICAST_TTL`
- `IP_ADD_MEMBERSHIP`
- `IP_DROP_MEMBERSHIP`

#### TCP options

TCP sockets only. Used with level argument `IPPROTO_TCP`.

- `TCP_NODELAY`

Options relating to TCP keepalive probes:

- `TCP_KEEPALIVE` (int value, TCP keepalive period in milliseconds)
- `TCP_KEEPIDLE` (same as `TCP_KEEPALIVE`, but the value is in seconds)
- `TCP_KEEPINTVL` (int value, interval between keepalive probes in seconds)
- `TCP_KEEPCNT` (int value, number of keepalive probes before timing out)
IPv6 options
IPv6 sockets only. Used with level argument IPPROTO_IPV6
- IPV6_CHECKSUM
- IPV6_V6ONLY

For multicast IPv6 UDP sockets:
- IPV6_JOIN_GROUP / IPV6_ADD_MEMBERSHIP
- IPV6_LEAVE_GROUP / IPV6_DROP_MEMBERSHIP
- IPV6_MULTICAST_IF
- IPV6_MULTICAST_HOPS
- IPV6_MULTICAST_LOOP

fcntl

The fcntl() function is a standard API for manipulating options related to a file descriptor. In ESP-IDF, the virtual file system layer is used to implement this function.

When the file descriptor is a socket, only the following fcntl() values are supported:
- O_NONBLOCK to set/clear non-blocking I/O mode. Also supports O_NDELAY, which is identical to O_NONBLOCK.
- O_RDONLY, O_WRONLY, O_RDWR flags for different read/write modes. These can read via F_GETFL only, they cannot be set using F_SETFL. A TCP socket will return a different mode depending on whether the connection has been closed at either end or is still open at both ends. UDP sockets always return O_RDWR.

ioctl

The ioctl() function provides a semi-standard way to access some internal features of the TCP/IP stack. In ESP-IDF, the virtual file system layer is used to implement this function.

When the file descriptor is a socket, only the following ioctl() values are supported:
- FIONREAD returns the number of bytes of pending data already received in the socket’s network buffer.
- FIONBIO is an alternative way to set/clear non-blocking I/O status for a socket, equivalent to fcntl(fd, F_SETFL, O_NONBLOCK, ...).

4.19.3 Netconn API

lwIP supports two lower level APIs as well as the BSD Sockets API: the Netconn API and the Raw API.

The lwIP Raw API is designed for single threaded devices and is not supported in ESP-IDF.

The Netconn API is used to implement the BSD Sockets API inside lwIP, and it can also be called directly from ESP-IDF apps. This API has lower resource usage than the BSD Sockets API, in particular it can send and receive data without needing to first copy it into internal lwIP buffers.

Important: Espressif does not test the Netconn API in ESP-IDF. As such, this functionality is enabled but not supported. Some functionality may only work correctly when used from the BSD Sockets API.

For more information about the Netconn API, consult lwip/lwip/src/include/lwip/api.h and this wiki page which is part of the unofficial lwIP Application Developers Manual.

4.19.4 lwIP FreeRTOS Task

lwIP creates a dedicated TCP/IP FreeRTOS task to handle socket API requests from other tasks.

A number of configuration items are available to modify the task and the queues (“mailboxes”) used to send data to/from the TCP/IP task.
4.19.5 IPv6 Support

Both IPv4 and IPv6 are supported as dual stack and enabled by default (IPv6 may be disabled if it’s not needed, see Minimum RAM usage). IPv6 support is limited to Stateless Autoconfiguration only. Stateful configuration is not supported in ESP-IDF (not in upstream lwip). IPv6 Address configuration is defined by means of these protocols or services:

- SLAAC IPv6 Stateless Address Autoconfiguration (RFC-2462)
- DHCPv6 Dynamic Host Configuration Protocol for IPv6 (RFC-8415)

None of these two types of address configuration is enabled by default, so the device uses only Link Local addresses or statically defined addresses.

Stateless Autoconfiguration Process

To enable address autoconfiguration using Router Advertisement protocol please enable:

- `CONFIG_LWIP_IPV6_AUTOCONFIG`

This configuration option enables IPv6 autoconfiguration for all network interfaces (in contrast to the upstream lwIP, where the autoconfiguration needs to be explicitly enabled for each netif with `netif->ip6_autoconfig_enabled=1`).

DHCPv6

DHCPv6 in lwIP is very simple and support only stateless configuration. It could be enabled using:

- `CONFIG_LWIP_IPV6_DHCP6`

Since the DHCPv6 works only in its stateless configuration, the Stateless Autoconfiguration Process has to be enabled, too, by means of `CONFIG_LWIP_IPV6_AUTOCONFIG`. Moreover, the DHCPv6 needs to be explicitly enabled form the application code using

```c
dhcp6_enable_stateless(netif);
```

DNS servers in IPv6 autoconfiguration

In order to autoconfigure DNS server(s), especially in IPv6 only networks, we have these two options

- Recursive domain name system – this belongs to the Neighbor Discovery Protocol (NDP), uses Stateless Autoconfiguration Process. Number of servers must be set `CONFIG_LWIP_IPV6_RDNSS_MAX_DNS_SERVERS`, this is option is disabled (set to 0) by default.
- DHCPv6 stateless configuration – uses `DHCPv6` to configure DNS servers. Note that the this configuration assumes IPv6 Router Advertisement Flags (RFC-5175) to be set to
  - Managed Address Configuration Flag = 0
  - Other Configuration Flag = 1

4.19.6 esp-lwip custom modifications

Additions

The following code is added which is not present in the upstream lwIP release:
Thread-safe sockets  It is possible to close() a socket from a different thread to the one that created it. The close() call will block until any function calls currently using that socket from other tasks have returned.

It is, however, not possible to delete a task while it is actively waiting on select() or poll() APIs. It is always necessary that these APIs exit before destroying the task, as this might corrupt internal structures and cause subsequent crashes of the lwIP. (These APIs allocate globally referenced callback pointers on stack, so that when the task gets destroyed before unrolling the stack, the lwIP would still hold pointers to the deleted stack)

On demand timers  lwIP IGMP and MLD6 features both initialize a timer in order to trigger timeout events at certain times.

The default lwIP implementation is to have these timers enabled all the time, even if no timeout events are active. This increases CPU usage and power consumption when using automatic light sleep mode. esp-lwip default behaviour is to set each timer “on demand” so it is only enabled when an event is pending.

To return to the default lwIP behaviour (always-on timers), disable CONFIG_LWIP_TIMERS_ONDEMAND.

Lwip timers API  When users are not using WiFi, these APIs provide users with the ability to turn off LwIP timer to reduce power consumption.

The following API functions are supported. For full details see lwip/lwip/src/include/lwip/timeouts.h.

• sys_timeouts_init()
• sys_timeouts_deinit()

Additional Socket Options

• Some standard IPV4 and IPV6 multicast socket options are implemented (see Socket Options).
• Possible to set IPV6-only UDP and TCP sockets with IPV6_V6ONLY socket option (normal lwIP is TCP only).

IP layer features

• IPV4 source based routing implementation is different.
• IPV4 mapped IPV6 addresses are supported.

Customized lwIP hooks  The original lwIP supports implementing custom compile-time modifications via LWIP_HOOK_FILENAME. This file is already used by the IDF port layer, but IDF users could still include and implement any custom additions via a header file defined by the macro ESP_IDF_LWIP_HOOK_FILENAME. Here is an example of adding a custom hook file to the build process (the hook is called my_hook.h and located in the project’s main folder):

```c
idf_component_get_property(lwip lwip COMPONENT_LIB)
target_compile_options({lwip} PRIVATE "-I{PROJECT_DIR}/main")
target_compile_definitions({lwip} PRIVATE "-DESP_IDF_LWIP_HOOK_FILENAME="my_hook_.h")
```

Limitations

Calling send() or sendto() repeatedly on a UDP socket may eventually fail with errno equal to ENOMEM. This is a limitation of buffer sizes in the lower layer network interface drivers. If all driver transmit buffers are full then UDP transmission will fail. Applications sending a high volume of UDP datagrams who don’t wish for any to be dropped by the sender should check for this error code and re-send the datagram after a short delay.

Increasing the number of TX buffers in the Wi-Fi or Ethernet project configuration (as applicable) may also help.
4.19.7 Performance Optimization

TCP/IP performance is a complex subject, and performance can be optimized towards multiple goals. The default settings of ESP-IDF are tuned for a compromise between throughput, latency, and moderate memory usage.

Maximum throughput

Espressif tests ESP-IDF TCP/IP throughput using the wifi/iperf example in an RF sealed enclosure. The wifi/iperf/sdkconfig.defaults file for the iperf example contains settings known to maximize TCP/IP throughput, usually at the expense of higher RAM usage. To get maximum TCP/IP throughput in an application at the expense of other factors then suggest applying settings from this file into the project sdkconfig.

 AVC: Suggest applying changes a few at a time and checking the performance each time with a particular application workload.

- If a lot of tasks are competing for CPU time on the system, consider that the lwIP task has configurable CPU affinity (CONFIG_LWIP_TCP_TASK_AFFINITY) and runs at fixed priority ESP_TASK_TCP_IP_Prio (18). Configure competing tasks to be pinned to a different core, or to run at a lower priority. See also Built-In Task Priorities.
- If using select() function with socket arguments only, disabling CONFIG_VFS_SUPPORT_SELECT will make select() calls faster.
- If there is enough free IRAM, select CONFIG_LWIP_IRAM_OPTIMIZATION to improve TX/RX throughput.

If using a Wi-Fi network interface, please also refer to Wi-Fi缓冲区使用情况.

Minimum latency

Except for increasing buffer sizes, most changes which increase throughput will also decrease latency by reducing the amount of CPU time spent in lwIP functions.

- For TCP sockets, lwIP supports setting the standard TCP_NODELAY flag to disable Nagle’s algorithm.

Minimum RAM usage

Most lwIP RAM usage is on-demand, as RAM is allocated from the heap as needed. Therefore, changing lwIP settings to reduce RAM usage may not change RAM usage at idle but can change it at peak.

- Reducing CONFIG_LWIP_MAX_SOCKETS reduces the maximum number of sockets in the system. This will also cause TCP sockets in the WAIT_CLOSE state to be closed and recycled more rapidly (if needed to open a new socket), further reducing peak RAM usage.
- Reducing CONFIG_LWIP_TCP_RB_RECVMBX_SIZE, CONFIG_LWIP_TCP_RECVMBX_SIZE and CONFIG_LWIP_UDP_RECVMBX_SIZE reduce memory usage at the expense of throughput, depending on usage.
- Reducing CONFIG_LWIP_TCP_MSL, CONFIG_LWIP_TCP_FIN_WAIT_TIMEOUT reduces the maximum segment lifetime in the system. This will also cause TCP sockets in the TIME_WAIT, FIN_WAIT_2 state to be closed and recycled more rapidly.
- Disable CONFIG_LWIP_IPV6 can save about 39 KB for firmware size and 2KB RAM when system power up and 7KB RAM when TCP/IP stack running. If there is no requirement for supporting IPV6 then it can be disabled to save flash and RAM footprint.

If using Wi-Fi, please also refer to Wi-Fi缓冲区使用情況.

Peak Buffer Usage  The peak heap memory that lwIP consumes is the theoretically-maximum memory that the lwIP driver consumes. Generally, the peak heap memory that lwIP consumes depends on:

- the memory required to create a UDP connection: lwip_udp_conn
- the memory required to create a TCP connection: lwip_tcp_conn
Chapter 4. API

- the number of UDP connections that the application has: lwip_udp_con_num
- the number of TCP connections that the application has: lwip_tcp_con_num
- the TCP TX window size: lwip_tcp_tx_win_size
- the TCP RX window size: lwip_tcp_rx_win_size

So, the peak heap memory that the LwIP consumes can be calculated with the following formula:

\[
\text{lwip_dynamic_peak_memory} = (\text{lwip_udp_con_num} \times \text{lwip_udp_conn}) + (\text{lwip_tcp_con_num} \times (\text{lwip_tcp_tx_win_size} + \text{lwip_tcp_rx_win_size} + \text{lwip_tcp_conn}))
\]

Some TCP-based applications need only one TCP connection. However, they may choose to close this TCP connection and create a new one when an error (such as a sending failure) occurs. This may result in multiple TCP connections existing in the system simultaneously, because it may take a long time for a TCP connection to close, according to the TCP state machine (refer to RFC793).

4.20 存储器类型

ESP32 芯片具有不同类型的存储器和灵活的存储器映射特性，本小节将介绍 ESP-IDF 默认如何使用这些功能。

ESP-IDF 区分了指令总线（IRAM、IROM、RTC FAST memory）和数据总线 (DRAM、DROM)。指令存储器是可执行的，只能通过 4 字节对齐字读取或写入。数据存储器不可执行，可以通过单独的字节操作访问。有关总线的更多信息，请参阅 ESP32 技术参考手册 > 系统和存储器 [PDF]。

4.20.1 DRAM（数据 RAM）

非常量静态数据 (.data 段) 和零初始化数据 (.bss 段) 由链接器放入内部 SRAM 作为数据存储。此区域中的剩余空间可在程序运行时用作堆。

通过应用 EXTRAM_BSS_ATTR 宏，零初始化数据也可以放入外部 RAM，使用这个宏需要启用 CONFIG_SPIRAM_ALLOW_BSS_EXTERNAL_MEMORY，详情请见允许 .bss 段放入片外存储器。

如果使用蓝牙堆栈，内存 DRAM 区域的可用大小将减少 64 KB (由于起始地址移动到 0x3FFC0000)。如果使用内存跟踪功能，该区域的长度还会减少 16 KB 或 32 KB。由于 ROM 引起的一些内存碎片问题，不可能将所有可用的 DRAM 用于静态分配，但是剩余的 DRAM 在运行时仍可用作堆。

常量数据也可能被放人 DRAM，例如当它被用于 non-flash-safe ISR 时（具体请参考如何将代码放入 IRAM）。

“noinit” DRAM

可以将 __NOINIT_ATTR 宏用作属性，从而将数据放入 .noinit 部分。放入该部分的值在启动时不会被初始化，在软件重启后也会保持值不变。

通过使用 EXT_RAM_NOINIT_ATTR 宏，noinit 数据也可以放入外部 RAM 中。为此，需要启用 CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY，可参考允许 .noinit 段放入片外存储器。如果没有启用 CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY，EXT_RAM_NOINIT_ATTR 会和 __NOINIT_ATTR 一样，将数据放入内部 RAM 的 .noinit 部分。

示例:

```c
__NOINIT_ATTR uint32_t noinit_data;
```
4.20.2 IRAM（指令 RAM）

ESP-IDF 将内部 SRAM0 的部分区域分配为指令 RAM。可在 ESP32 技术参考手册 > 系统和存储器 > 内部存储器 [PDF] 中查看 IRAM 区域的定义。该内存中第一个 64 KB 块用于 PRO 和 APP MMU 缓存，其余部分（从 0x40080000 到 0x400A0000）用于存储需要从 RAM 运行的应用程序部分。

何时需要将代码放入 IRAM

以下情况时应将部分应用程序放入 IRAM:
- 如果在注册中断处理程序时使用了 ESP_INTR_FLAG_IRAM，则中断处理程序必须要放入 IRAM。更多信息可参考 IRAM 定义中断处理程序。
- 可将一些时序关键代码放入 IRAM，以减少从 flash 中加载代码造成的相关损失。ESP32 通过 MMU 缓存从 flash 中读取代码和数据。在某些情况下，将函数放入 IRAM 可以减少由缓存未命中造成的延迟，从而显著提高函数的性能。

如何将代码放入 IRAM

借助链接器脚本，一些代码会被自动放入 IRAM 区域中。

如果需要将某些特定的应用程序代码放入 IRAM，可以使用链接器脚本生成机制 功能。在组件中添加链接器脚本片段文件，该片段文件中，可以为输出目标源文件或其中的个别函数打上 noflash 标签。更多信息可参考链接器脚本生成机制。

或者，也可以通过使用 IRAM_ATTR 宏在源代码中指定需要放入 IRAM 的代码:

```c
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg) {
    // ...
}
```

放于 IRAM 后可能会导致 IRAM 安全中断处理程序出现问题：
- IRAM_ATTR 函数中的字符串或常量可能没有自动放入 RAM 中。这时可以使用 DRAM_ATTR 属性进行标记，或者也可以使用链接器脚本方法将它们放入 RAM 中。

```c
void IRAM_ATTR gpio_isr_handler(void* arg) {
    const static DRAM_ATTR uint8_t INDEX_DATA[] = { 45, 33, 12, 0 };
    const static char *MSG = DRAM_STR("I am a string stored in RAM");
}
```

注意，具体哪些数据需要被标记为 DRAM_ATTR 可能很难确定。如果没有被标记为 DRAM_ATTR，某些变量或表达式有时会被编译器视为常量（即使它们没有被标记为 const）并将其放入 flash 中。


4.20.3 IROM（代码从 flash 中运行）

如果一个函数没有被显式地声明放在 IRAM 或者 RTC 存储器中，则它会放在 flash 中。由于 IRAM 空间有限，应用程序的大部二进制代码都需要放入 IROM 中。

允许从 flash 中执行代码的 Flash MMU 机制可参考 ESP32 技术参考手册 > 存储器管理和保护单元 (MMU, MPU) [PDF]。
在启动过程中，从 IRAM 中运行的引导加载程序配置 MMU flash 缓存，将应用程序的指令代码区域映射到指令空间。通过 MMU 访问的 flash 使用一些内部 SRAM 进行缓存，访问缓存的 flash 数据与访问其它类型的内部存储器一样快。

### 4.20.4 DROM（数据存储在 flash 中）

默认情况下，链接器将常量数据放入一个映射到 MMU flash 缓存的区域中。这与 IROM（代码从 flash 中运行）部分相同，但此处用于只读数据而不是可执行代码。

唯一没有默认放入 DROM 的常量数据是被编译器嵌入到应用程序代码中的字面常量。这些被放置在周围函数的可执行指令中。

DRAM_ATTR 属性可以用来强制将常量从 DRAM 放入 DRAM（数据 RAM）部分（见上文）。

### 4.20.5 RTC Slow memory（RTC 慢速存储器）

从 RTC 存储器运行的代码使用的全局和静态变量必须放入 RTC Slow memory 中。例如深度睡眠变量可以在放在 RTC Slow memory 中，而不是 RTC FAST memory，或者也可以放入由 ULP 协处理器编程访问的代码和变量。

RTC_NOINIT_ATTR 属性宏可以用来将数据放入 RTC Slow memory。放入此类型存储器的值从深度睡眠模式中醒来后会保持值不变。

示例:

```c
RTC_NOINIT_ATTR uint32_t rtc_noinit_data;
```

### 4.20.6 RTC FAST memory（RTC 快速存储器）

RTC FAST memory 的同一区域既可以作为指令存储器也可以作为数据存储器进行访问。从深度睡眠模式唤醒后必须要运行的代码要放在 RTC 存储器中，更多信息请查阅文档深度睡眠。

RTC FAST memory 只可以被 PRO CPU 访问。

在单核模式下，除非禁用 CONFIG_ESP_SYSTEM_ALLOW_RTC_FAST_MEM_AS_HEAP 选项，否则剩余的 RTC FAST memory 会被添加到堆中。该部分内存可以和 DRAM（数据 RAM）互换使用，但是访问速度稍慢，且不具备 DMA 功能。

### 4.20.7 具备 DMA 功能

大多数的 DMA 控制器（比如 SPI，sdmmc 等）都要求发送/接收缓冲区放在 DRAM 中，并且按字对齐。我们建议将 DMA 缓冲区放在静态变量而不是堆栈中。使用 DMA_ATTR 宏可以声明该全局/本地的静态变量具备 DMA 功能，例如:

```c
DMA_ATTR uint8_t buffer[] = "I want to send something";

void app_main()
{
    // 初始化代码
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8 * sizeof(buffer),
    };
    spi_device_transmit(spi, temp);
    // 其它程序
}
```

或者:
void app_main()
{
    DMA_ATTR static uint8_t buffer[] = "I want to send something";
    // 初始化代码
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8 * sizeof(buffer),
    };
    spi_device_transmit(spi, &temp);
    // 其它程序
}

也可以通过使用 MALLOC_CAP_DMA 标志来动态分配具备 DMA 能力的内存缓冲区。

### 4.20.8 在堆栈中放置 DMA 缓冲区

可以在堆栈中放置 DMA 缓冲区，但建议尽量避免。如果实在有需要的话，请注意以下几点：

- 如果堆栈在 PSRAM 中，则不建议将 DRAM 缓冲区放在堆栈上。如果任务堆栈在 PSRAM 中，则必须执行**例外 RAM** 中描述的几个步骤。
- 在函数中使用 **WORD_ALIGNED_ATTR** 宏来修饰变量，将其放在适当的位置上，比如:

```c
void app_main()
{
    uint8_t stuff;
    WORD_ALIGNED_ATTR uint8_t buffer[] = "I want to send something";  // 声明...
    // 初始化代码
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8 * sizeof(buffer),
    };
    spi_device_transmit(spi, &temp);
    // 其它程序
}
```

### 4.21 OpenThread

OpenThread is a IP stack running on the 802.15.4 MAC layer which features mesh network and low power consumption.

#### 4.21.1 Mode of the OpenThread stack

OpenThread can run under the following modes on Espressif chips:

**Standalone node**

The full OpenThread stack and the application layer runs on the same chip. This mode is available on chips with 15.4 radio such as ESP32-H4.
Radio Co-Processor (RCP)

The chip will be connected to another host running the OpenThread IP stack. It will send and received 15.4 packets on behalf of the host. This mode is available on chips with 15.4 radio such as ESP32-H4. The underlying transport between the chip and the host can be SPI or UART. For sake of latency, we recommend to use SPI as the underlying transport.

OpenThread host

For chips without 15.4 radio, it can be connected to an RCP and run OpenThread under host mode. This mode enables OpenThread on Wi-Fi chips such as ESP32, ESP32-S2, ESP32-S3 and ESP32-C3. The following diagram shows how devices work under different modes:

![Diagram showing OpenThread device modes](image)

**4.21.2 How To Write an OpenThread Application**

The OpenThread openthread/ot_cli example will be a good place to start at. It demonstrates basic OpenThread initialization and simple socket-based server and client.

**Before OpenThread initialization**

- s1.1 The main task calls `esp_vfs_eventfd_register()` to initialize the eventfd virtual filesystem. The eventfd file system is used for task notification in the OpenThread driver.
- s1.2 The main task calls `nvs_flash_init()` to initialize the NVS where the Thread network data is stored.
- s1.3 Optional. The main task calls `esp_netif_init()` only when it wants to create the network interface for Thread.
- s1.4: The main task calls `esp_event_loop_create()` to create the system Event task and initialize an application event’s callback function.

**OpenThread stack initialization**

- s2.1: Call `esp_openthread_init()` to initialize the OpenThread stack.
OpenThread network interface initialization

The whole stage is optional and only required if the application wants to create the network interface for Thread.
- s3.1: Call `esp_netif_new()` with `ESP_NETIF_DEFAULT_OPENTHREAD` to create the interface.
- s3.2: Call `esp_openthread_netif_glue_init()` to create the OpenThread interface handlers.
- s3.3: Call `esp_netif_attach()` to attach the handlers to the interface.

The OpenThread main loop

- s4.3: Call `esp_openthread_launch_mainloop()` to launch the OpenThread main loop. Note that this is a busy loop and will not return until the OpenThread stack is terminated.

Calling OpenThread APIs

The OpenThread APIs are not thread-safe. When calling OpenThread APIs from other tasks, make sure to hold the lock with `esp_openthread_lock_acquire()` and release the lock with `esp_openthread_lock_release()` afterwards.

Deinitialization

The following steps are required to deinitialize the OpenThread stack: - Call `esp_netif_destroy()` and `esp_openthread_netif_glue_deinit()` to deinitialize the OpenThread network interface if you have created one. - Call `esp_openthread_deinit()` to deinitialize the OpenThread stack.

4.21.3 The OpenThread border router

The OpenThread border router connects the Thread network with other IP networks. It will provide IPv6 connectivity, service registration and commission functionality. To launch an OpenThread border router on an ESP chip, you need to connect an RCP to a Wi-Fi capable chip such as ESP32. Call `esp_openthread_border_router_init()` during the initialization will launch all the border routing functionalities.

You may refer to the `openthread/ot_br` example and the README for further border router details.

4.22 分区表

4.22.1 概述

每片 ESP32 的 flash 可以包含多个应用程序，以及多种不同类型的数 (例如校准数据、文件系统数据、参数存储数据等)。因此，我们在 flash 的默认偏移地址 0x8000 处烧写一张分区表。

分区表的长度为 0x0C00 字节（最多可以保存 95 条分区表条目）。分区表数据后还保存着该表的 MD5 校验和，用于验证分区表的完整性。此外，如果芯片使能了安全启动功能，则该分区表后还会保存签名信息。分区表中的每个条目都包括以下几个部分：Name（标签）、Type（app, data 等）、SubType 以及在 flash 中的偏移量（分区的加载地址）。

在使用分区表时，最简单的方法就是打开项目配置菜单（idf.py menuconfig），并在 `CONFIG_PARTITION_TABLE_TYPE` 下选择一个预定义的分区表：
- “Single factory app, no OTA”
- “Factory app, two OTA definitions”

在以上两种选项中，出厂应用程序均将被烧录至 flash 的 0x10000 偏移地址处。这时，运行 `idf.py partition-table`，即可打印当前使用分区表的信息摘要。
4.22.2 内置分区表

以下是“Single factory app, no OTA”选项的分区表信息摘要:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>SubType</th>
<th>Offset</th>
<th>Size</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvs</td>
<td>data</td>
<td>nvs</td>
<td>0x9000</td>
<td>0x6000</td>
<td></td>
</tr>
<tr>
<td>phy_init</td>
<td>data</td>
<td>phy</td>
<td>0xf000</td>
<td>0x1000</td>
<td></td>
</tr>
<tr>
<td>factory</td>
<td>app</td>
<td>factory</td>
<td>0x10000</td>
<td>1M</td>
<td></td>
</tr>
</tbody>
</table>

- flash 的 0x10000 (64 KB) 偏移地址处存放一个标记为“factory”的二进制应用程序，且启动加载器将默认加载这个应用程序。
- 分区表中定义了两个数据区域，分别用于存储 NVS 库专用分区和 PHY 初始化数据。

以下是“Factory app, two OTA definitions”选项的分区表信息摘要:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>SubType</th>
<th>Offset</th>
<th>Size</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvs</td>
<td>data</td>
<td>nvs</td>
<td>0x9000</td>
<td>0x4000</td>
<td></td>
</tr>
<tr>
<td>otsdata</td>
<td>data</td>
<td>ots</td>
<td>0xd000</td>
<td>0x2000</td>
<td></td>
</tr>
<tr>
<td>phy_init</td>
<td>data</td>
<td>phy</td>
<td>0xf000</td>
<td>0x1000</td>
<td></td>
</tr>
<tr>
<td>factory</td>
<td>app</td>
<td>factory</td>
<td>0x10000</td>
<td>1M</td>
<td></td>
</tr>
<tr>
<td>ota_0</td>
<td>app</td>
<td>ota_0</td>
<td>0x110000</td>
<td>1M</td>
<td></td>
</tr>
<tr>
<td>ota_1</td>
<td>app</td>
<td>ota_1</td>
<td>0x210000</td>
<td>1M</td>
<td></td>
</tr>
</tbody>
</table>

- 分区表中定义了三个应用程序分区，这三个分区的类型都被设置为“app”，但具体 app 类型不同。其中，位于 0x10000 偏移地址处的为出厂应用程序（factory），其余两个为 OTA 应用程序（ota_0, ota_1）。
- 新增了一个名为“otadata”的数据分区，用于保存 OTA 升级时需要的数据。启动加载器会查询该分区的数据，以判断该从哪个 OTA 应用程序分区加载程序。如果“otadata”分区为空，则会执行出厂程序。

4.22.3 创建自定义分区表

如果在 menuconfig 中选择了“Custom partition table CSV”，则还需要输入该分区表的 CSV 文件在项目中的路径。CSV 文件可以根据需要，描述任意数量的分区信息。

CSV 文件的格式与上面摘要中打印的格式相同，但是 CSV 文件中并非所有字段都是必需的。例如下面是一个自定义的 OTA 分区表的 CSV 文件:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>SubType</th>
<th>Offset</th>
<th>Size</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvs</td>
<td>data</td>
<td>nvs</td>
<td>0x9000</td>
<td>0x4000</td>
<td></td>
</tr>
<tr>
<td>otsdata</td>
<td>data</td>
<td>ots</td>
<td>0xd000</td>
<td>0x2000</td>
<td></td>
</tr>
<tr>
<td>phy_init</td>
<td>data</td>
<td>phy</td>
<td>0xf000</td>
<td>0x1000</td>
<td></td>
</tr>
<tr>
<td>factory</td>
<td>app</td>
<td>factory</td>
<td>0x10000</td>
<td>1M</td>
<td></td>
</tr>
<tr>
<td>ota_0</td>
<td>app</td>
<td>ota_0</td>
<td></td>
<td></td>
<td>1M</td>
</tr>
<tr>
<td>ota_1</td>
<td>app</td>
<td>ota_1</td>
<td></td>
<td></td>
<td>1M</td>
</tr>
<tr>
<td>nvs_key</td>
<td>data</td>
<td>nvs_keys</td>
<td></td>
<td></td>
<td>0x1000</td>
</tr>
</tbody>
</table>

- 字段之间的空格会被忽略，任何以 # 开头的行（注释）也会被忽略。
- CSV 文件中的每个非注释行均为一个分区定义。
- 每个分区的 Offset 字段可以为空，gen_esp32part.py 工具会从分区表位置的后面开始自动计算并填充该分区的偏移地址，同时确保每个分区的偏移地址正确对齐。

Name 字段

Name 字段可以是任何有意义的名称，但不能超过 16 个字节，其中包括一个空字节（之后的内容将被截断）。该字段对 ESP32 并不是特别重要。
Chapter 4. API 指南

**Type 字段**

Type 字段可以指定为 app (0x00) 或者 data (0x01)，也可以直接使用数字 0-254（或者十六进制 0x00-0xFE)。注意，0x00-0x3F 不得使用（留给 esp-idf 的核心功能）。

如果您的应用程序需要以 ESP-IDF 尚未支持的格式存储数据，请在 0x40-0xFE 内添加一个自定义分区类型。

参考 esp_partition_type_t 关于 app 和 data 分区的枚举定义。

如果用 C++ 编写，那么指定一个应用程序定义的分区类型，需要在 esp_partition_type_t 中使用整数，从而与函数 API 一起使用。例如:

```c
static const esp_partition_type_t APP_PARTITION_TYPE_A = {esp_partition_type_t...
```

注意，启动加载器将忽略 app (0x00) 和 data (0x01) 以外的其他分区类型。

**SubType 字段**

SubType 字段长度为 8 bit，内容与具体分区 Type 有关。目前，esp-idf 仅仅规定了“app”和“data”两种分区类型的子类型含义。

参考 esp_partition_subtype_t，以了解 ESP-IDF 定义的全部子类型列表，包括:

- 当 Type 定义为 app 时，SubType 字段可以指定为 factory (0x00)、ota_0 (0x10) ... ota_15 (0x1F) 或者 test (0x20)。
  - factory (0x00) 是默认的 app 分区。启动加载器将默认加载该应用程序。但如果存在类型为 data/ota 分区，则启动加载器将加载 data/ota 分区中的数据，进而启动该各个OTA镜像文件。
  - OTA 升级永远都不会更新 factory 分区中的内容。
  - 如果您希望在 OTA 项目中预留更多 flash，可以删除 factory 分区，转而使用 ota_0 分区。
  - ota_0 (0x10) ... ota_15 (0x1F) 为 OTA 应用程序分区。启动加载器将根据 OTA 数据分区中的数据来决定加载哪个 OTA 应用程序分区中的程序。在使用 OTA 功能时，应用程序应至少拥有 2 个 OTA 应用程序分区 (ota_0 和 ota_1)。更多详细信息，请参考 OTA 文档。
  - test (0x20) 为预留的子类型，用于工厂测试流程。如果没有其他有效 app 分区，test 将作为选启动分区使用。也可以配置启动加载器在每次启动时读取 GPIO，如果 GPIO 被拉低则启动该分区。详细信息请查阅从测试固件启动。
- 当 Type 定义为 data 时，SubType 字段可以指定为 ota (0x00)、phy (0x01)、nvs (0x02)、nvs_keys (0x04) 或者其他组件特定的子类型（请参考子类型枚举）。
  - ota (0) 即 OTA 数据分区，用于存储当前所选的 OTA 应用程序的信息。这个分区的大小需要设定为 0x2000，更多详细信息，请参考 OTA 文档。
  - phy (1) 分区用于存放 PHY 初始化数据，从而保证可以为每个设备单独配置 PHY，而非必须采用固件中的统一 PHY 初始化数据。
    - 默认配置下，phy 分区并不启用，而是直接将 phy 初始化数据编译至应用程序中，从而节省分区表空间（直接将此分区删除）。
    - 如果需要从此分区加载 phy 初始化数据，请打开项目配置菜单 (idf.py menuconfig)，并且使能 CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION 选项。此时，您还需要手动将 phy 初始化数据烧至 flash (esp-idf 编译系统并不会自动完成该操作)。
  - nvs (2) 是专门给非易失性存储 (NVS API) 使用的分区。
    - 用于存储每台设备的 PHY 校准数据（注意，并不是 PHY 初始化数据）。
    - 用于存储 Wi-Fi 数据（如果使用了 esp_wifi_set_storage(WIFI_STORAGE_FLASH) 初始化函数）。
    - NVS API 还可以用于其他应用程序数据。
    - 强烈建议您应为 NVS 分区分配至少 0x3000 字节空间。
    - 如果使用 NVS API 存储大量数据，请增加 NVS 分区的大小（默认是 0x6000 字节）。
  - nvs_keys (4) 是 NVS 秘钥分区。详细信息，请参考非易失性存储 (NVS API) 文档。
    - 用于存储加密密钥（如果启用了 NVS 加密功能）。
    - 此分区应至少设定为 0x0006 字节。
Chapter 4. API 指南

- ESP-IDF 还支持其它预定义的子类型用于数据存储，包括 FAT 文件系统 (ESP_PARTITION_SUBTYPE_DATA_FAT), SPIFFS (ESP_PARTITION_SUBTYPE_DATA_SPIFFS) 等。

包含的数据子类型已预留给 esp-idf 未来使用。
- 如果分区类型是由应用程序定义的任意值（0x40-0xFE），那么 subtype 字段可以是由应用程序选择的任何值（0x00-0xFE）。

请注意如果用 C++ 编写，应用程序定义的子类型值需要转换为 esp_partition_type_t，从而与分区 API 一起使用。

额外分区 SubType 字段

组件可以通过设置 EXTRA_PARTITION_SUBTYPES 属性来定义额外的分区子类型。EXTRA_PARTITION_SUBTYPES 是一个 CMake 列表，其中的每个条目由字符串组成，以逗号为分隔，格式为 <type>, <subtype>, <value>。构建系统通过该属性会自动添加额外的子类型，并在 esp_partition_subtype_t 中插入名为 ESP_PARTITION_SUBTYPE_<type>_<subtype> 的字段。项目可以使用这个子类型来定义分区表 CSV 文件中的分区，并使用 esp_partition_subtype_t 中的新字段。

Offset 和 Size 字段

分区若偏移地址为空，则会紧跟着前一个分区之后开始；若为首个分区，则将紧跟着分区表开始。

app 分区的偏移地址必须要与 0x10000 (64K) 对齐。如果将偏移字段留空。gen.esp32part.py 工具会自动计算得到一个满足对齐要求的偏移地址。如果 app 分区的偏移地址没有与 0x10000 (64K) 对齐，则该工具会报错。

app 分区的大小和偏移地址可以采用十进制数，以 0x 为前缀的十六进制数，且支持 K 或 M 的倍数单位（分别代表 1024 和 1024*1024 字节）。

如果您希望允许分区表中的分区采用任意起始偏移量 (CONFIG_PARTITION_TABLE_OFFSET)，请将分区表（CSV 文件）中所有分区的偏移字段都留空。注意，此时，如果您更改了分区表中任意分区的偏移地址，其他分区的偏移地址也会跟着改变。这种情况下，如果您之前还曾设定某个分区采用固定偏移地址，则可能造成分区表冲突，从而导致报错。

Flags 字段

当前仅支持 encrypted 标记。如果 Flags 字段设置为 encrypted，且已启用 Flash 功能，则该分区将会被加密。

备注：app 分区始终会被加密，不管 Flags 字段是否设置。

### 4.22.4 生成二进制分区表

编译到 ESP32 中的分区表采用二进制格式，而不是 CSV 文件本身。此时，partition_table/gen.esp32part.py 工具可以实现 CSV 和二进制文件之间的转换。

如果您在配置菜单（idf.py menuconfig）中设置了分区表 CSV 文件的名称，然后构建项目或执行 idf.py partition-table，这时，转换将在编译过程中自动完成。

手动将 CSV 文件转换为二进制文件:

```
python gen.esp32part.py input_partitions.csv binary_partitions.bin
```

手动将二进制文件转换为 CSV 文件:
Chapter 4. API 指南

```python gen_esp32part.py binary_partitions.bin input_partitions.csv```

在标准输出 (stdout) 上，打印二进制分区表的内容（运行 idf.py partition-table 时展示的信息摘要也是这样生成的）:

```python gen_esp32part.py binary_partitions.bin```

### 4.22.5 分区大小检查

ESP-IDF 构建系统将自动检查生成的二进制文件大小与可用的分区大小是否匹配，如果二进制文件太大，则会构建失败并报错。

目前会对以下二进制文件进行检查:

- 引导加载程序的二进制文件的大小要适合分区表前的区域大小（分区表前的区域都分配给了引导加载程序），具体请参考引导加载程序大小。
- 应用程序二进制文件应至少适合一个“app”类型的分区。如果不适合任何应用程序分区，则会构建失败。如果只适合某些应用程序分区，则会打印相关警告。

**备注**: 即使分区大小检查返回错误并导致构建失败，仍然会生成可以烧录的二进制文件（它们对于可用空间来说过大，因此无法正常工作）。

### MD5 校验和

二进制格式的分区表中含有一个 MD5 校验和。这个 MD5 校验和是根据分区表内容计算的，可在设备启动阶段，用于验证分区表的完整性。

用户可通过 gen_esp32part.py 的 --disable-md5sum 选项或者 CONFIG_PARTITION_TABLE_MD5 选项关闭 MD5 校验。对于 ESP-IDF v3.1 版本前的引导加载程序, 因为它不支持 MD5 校验, 所以无法正常启动并报告 invalid magic number 0xebeb, 此时用户可以使用此选项关闭 MD5 校验。

### 4.22.6 烧写分区表

- idf.py partition-table-flash: 使用 esptool.py 工具烧写分区表。
- idf.py flash: 会烧写所有内容，包括分区表。

在执行 idf.py partition-table 命令时，手动烧写分区表的命令也将打印在终端上。

**备注**: 分区表的更新并不会擦除根据旧分区表存储的数据。此时，您可以使用 idf.py erase-flash 命令或者 esptool.py erase_flash 命令来擦除 flash 中的所有内容。

### 4.22.7 分区工具 (parttool.py)

parttion_table 组件中有分区工具 parttool.py，可以在目标设备上完成分区相关操作。该工具具有如下用途:

- 读取分区，将内存存储到文件中 (read_partition)
- 将文件中的内容写入分区 (write_partition)
- 擦除分区 (erase_partition)
- 检索特定分区的名称、偏移、大小和 flag（“加密”）标志等信息 (get_partition_info)

用户若想通过编程方式完成相关操作，可从另一个 Python 脚本导入并使用分区工具，或者从 Shell 脚本调用分区工具。前者可使用工具的 Python API，后者可使用命令行界面。
Python API

首先请确保已导入 `parttool` 模块。

```python
import sys
import os

idf_path = os.environ["IDF_PATH"]  # 从环境中获取 IDF_PATH 的值
parttool_dir = os.path.join(idf_path, "components", "partition_table")  # parttool.py 位于 $IDF_PATH/components/partition_table 下

sys.path.append(parttool_dir)  # 使能 Python 寻找 parttool 模块
from parttool import *  # 导入 parttool 模块内的所有名称
```

要使用分区工具的 Python API，第一步是创建 `ParttoolTarget`:

```python
# 创建 parttool.py 的目标设备，并将目标设备连接到串行端口 /dev/ttyUSB1
target = ParttoolTarget("/dev/ttyUSB1")
```

现在，可使用创建的 `ParttoolTarget` 在目标设备上完成操作:

```python
# 擦除名为 ‘storage’ 的分区
target.erase_partition(PartitionName("storage"))

# 读取类型为 ‘data’、子类型为 ‘spiffs’ 的分区，保存至文件 ‘spiffs.bin’
target.read_partition(PartitionType("data", "spiffs"), "spiffs.bin")

# 将 ‘factory.bin’ 文件的内容写入 ‘factory’ 分区
target.write_partition(PartitionName("factory"), "factory.bin")

# 打印默认启动分区的大小
storage = target.get_partition_info(PARTITION_BOOT_DEFAULT)
print(storage.size)
```

使用 `PartitionName`、`PartitionType` 或 `PARTITION_BOOT_DEFAULT` 指定要操作的分区。顾名思义，这三个参数可以指向拥有特定名称的分区、特定类型和子类型的分区或默认启动分区。

更多关于 Python API 的信息，请查看分区工具的代码注释。

命令行界面

`parttool.py` 的命令行界面具有如下结构:

```
parttool.py [command-args] [subcommand] [subcommand-args]
```

- `command-args` - 执行主命令（parttool.py）所需的实际参数，多与目标设备有关
- `subcommand` - 要执行的操作
- `subcommand-args` - 所选操作的实际参数

```bash
# 擦除名为 ‘storage’ 的分区
parttool.py --port "/dev/ttyUSB1" erase_partition --partition-name=storage

# 读取类型为 ‘data’、子类型为 ‘spiffs’ 的分区，保存到 ‘spiffs.bin’ 文件
parttool.py --port "/dev/ttyUSB1" read_partition --partition-type=data --partition-
 subtype=spiffs --output "spiffs.bin"

# 将 ‘factory.bin’ 文件中的内容写入到 ‘factory’ 分区
parttool.py --port "/dev/ttyUSB1" write_partition --partition-name=factory --input
 "factory.bin"
```

(下页继续)
Chapter 4. API

4.23 Performance

ESP-IDF ships with default settings that are designed for a trade-off between performance, resource usage, and available functionality.

These guides describe how to optimize a firmware application for a particular aspect of performance. Usually this involves some trade-off in terms of limiting available functions, or swapping one aspect of performance (such as execution speed) for another (such as RAM usage).

4.23.1 How to Optimize Performance

1. Decide what the performance-critical aspects of your application are (for example: a particular response time to a certain network operation, a particular startup time limit, particular peripheral data throughput, etc.).
2. Find a way to measure this performance (some methods are outlined in the guides below).
3. Modify the code and project configuration and compare the new measurement to the old measurement.
4. Repeat step 3 until the performance meets the requirements set out in step 1.

4.23.2 Guides

Maximizing Execution Speed

Overview Optimizing execution speed is a key element of software performance. Code that executes faster can also have other positive effects, like reducing overall power consumption. However, improving execution speed may have trade-offs with other aspects of performance such as Minimizing Binary Size.

Choose What To Optimize If a function in the application firmware is executed once per week in the background, it may not matter if that function takes 10 ms or 100 ms to execute. If a function is executed constantly at 10 Hz, it matters greatly if it takes 10 ms or 100 ms to execute.

Most application firmwares will only have a small set of functions which require optimal performance. Perhaps those functions are executed very often, or have to meet some application requirements for latency or throughput. Optimization efforts should be targeted at these particular functions.

Measuring Performance The first step to improving something is to measure it.

Basic Performance Measurements If measuring performance relative to an external interaction with the world, you may be able to measure this directly (for example see the examples wifi/iperf and ethernet/iperf for measuring general network performance, or you can use an oscilloscope or logic analyzer to measure timing of an interaction with a device peripheral.)

Otherwise, one way to measure performance is to augment the code to take timing measurements:

```bash
parttool.py --port "/dev/ttyUSB1" get_partition_info --partition-boot-default --info size
```

更多信息可用 -help 指令查看：

```bash
parttool.py --help
```

```bash
parttool.py [subcommand] --help
```
#include "esp_timer.h"

void measure_important_function(void) {
    const unsigned MEASUREMENTS = 5000;
    uint64_t start = esp_timer_get_time();
    for (int retries = 0; retries < MEASUREMENTS; retries++) {
        important_function(); // This is the thing you need to measure
    }
    uint64_t end = esp_timer_get_time();
    printf("%u iterations took %llu milliseconds (%llu microseconds per...
        invocation), MEASUREMENTS, (end - start)/1000, (end - start)/MEASUREMENTS);
}

Executing the target multiple times can help average out factors like RTOS context switches, overhead of measurements, etc.

- Using `esp_timer_get_time()` generates “wall clock” timestamps with microsecond precision, but has moderate overhead each time the timing functions are called.
- It’s also possible to use the standard Unix `gettimeofday()` and `utime()` functions, although the overhead is slightly higher.
- Otherwise, including `hal/cpu_hal.h` and calling the HAL function `cpu_hal_get_cycle_count()` will return the number of CPU cycles executed. This function has lower overhead than the others. It is good for measuring very short execution times with high precision.
- The CPU cycles are counted per-core, so only use this method from an interrupt handler, or a task that is pinned to a single core.
- If making “microbenchmarks” (i.e. benchmarking only a very small routine of code that runs in less than 1-2 milliseconds) then flash cache performance can sometimes cause big variations in timing measurements depending on the binary. This happens because binary layout can cause different patterns of cache misses in a particular sequence of execution. If the test code is larger then this effect usually averages out. Executing a small function multiple times when benchmarking can help reduce the impact of flash cache misses. Alternatively, move this code to IRAM (see Targeted Optimizations).

**External Tracing**  
The 应用层跟踪库 allows measuring code execution with minimal impact on the code itself.

**Tasks**  
If the option `CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS` is enabled then the FreeRTOS API `vTaskGetRunTimeStats()` can be used to retrieve runtime information about the processor time used by each FreeRTOS task.

**SEGGER SystemView** is an excellent tool for visualizing task execution and looking for performance issues or improvements in the system as a whole.

**Improving Overall Speed**  
The following optimizations will improve the execution of nearly all code - including boot times, throughput, latency, etc:

- Set `CONFIG_ESPTOOLPY_FLASHFREQ` to 80 MHz. This is double the 40 MHz default value and will double the speed at which code is loaded or executed from flash. You should verify that the board or module that connects the ESP32 to the flash chip is rated for 80 MHz operation at the relevant temperature ranges, before changing this setting. The hardware datasheet(s) will have this information.
- Set `CONFIG_ESPTOOLPY_FLASHMODE` to QIO or QOUT mode (Quad I/O). Both will almost double the speed at which code is loaded or executed from flash compared to the default DIO mode. QIO is slightly faster than QOUT if both are supported. Note that both the flash chip model and the electrical connections between the ESP32 and the flash chip must support quad I/O modes or the SoC will not work correctly.
• Set `CONFIG_COMPILER_OPTIMIZATION` to “Optimize for performance (-O2)”. This may slightly increase binary size compared to the default setting, but will almost certainly increase performance of some code. Note that if your code contains C or C++ Undefined Behaviour then increasing the compiler optimization level may expose bugs that otherwise are not seen.

• If the application uses PSRAM and is based on ESP32 rev. 3 (ECO3), setting `CONFIG_ESP32_REV_MIN` to 3 will disable PSRAM bug workarounds, reducing the code size and improving overall performance.

• Avoid using floating point arithmetic (`float`). Even though ESP32 has a single precision hardware floating point unit, floating point calculations are always slower than integer calculations. If possible then use fixed point representations, a different method of integer representation, or convert part of the calculation to be integer only before switching to floating point.

• Avoid using double precision floating point arithmetic (`double`). These calculations are emulated in software and are very slow. If possible then use an integer-based representation, or single-precision floating point.

Reduce Logging Overhead  Although standard output is buffered, it’s possible for an application to be limited by the rate at which it can print data to log output once buffers are full. This is particularly relevant for startup time if a lot of output is logged, but can happen at other times as well. There are multiple ways to solve this problem:

• Reduce the volume of log output by lowering the app `CONFIG_LOG_DEFAULT_LEVEL` (the equivalent bootloader setting is `CONFIG_BOOTLOADER_LOG_LEVEL`). This also reduces the binary size, and saves some CPU time spent on string formatting.

• Increase the speed of logging output by increasing the `CONFIG_ESP_CONSOLE_UART_BAUDRATE`.

Not Recommended  The following options will also increase execution speed, but are not recommended as they also reduce the debuggability of the firmware application and may increase the severity of any bugs.

• Set `CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL` to disabled. This also reduces firmware binary size by a small amount. However, it may increase the severity of bugs in the firmware including security-related bugs. If necessary to do this to optimize a particular function, consider adding `#define NDEBUG` in the top of that single source file instead.

Targeted Optimizations  The following changes will increase the speed of a chosen part of the firmware application:

• Move frequently executed code to IRAM. By default, all code in the app is executed from flash cache. This means that it’s possible for the CPU to have to wait on a “cache miss” while the next instructions are loaded from flash. Functions which are copied into IRAM are loaded once at boot time, and then will always execute at full speed.

IRAM is a limited resource, and using more IRAM may reduce available DRAM, so a strategic approach is needed when moving code to IRAM. See IRAM（指令 RAM） for more information.

• Jump table optimizations can be re-enabled for individual source files that don’t need to be placed in IRAM. For hot paths in large switch cases this will improve performance. For instructions on how to add the `-fjump-tables -ftree-switch-conversion` options when compiling individual source files, see 算法编辑控制

Improving Startup Time  In addition to the overall performance improvements shown above, the following options can be tweaked to specifically reduce startup time:

• Minimizing the `CONFIG_LOG_DEFAULT_LEVEL` and `CONFIG_BOOTLOADER_LOG_LEVEL` has a large impact on startup time. To enable more logging after the app starts up, set the `CONFIG_LOG_MAXIMUM_LEVEL` as well and then call `esp_log_level_set()` to restore higher level logs. The `system/startup_time` main function shows how to do this.
• If using deep sleep, setting `CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP` allows a faster wake from sleep. Note that if using Secure Boot this represents a security compromise, as Secure Boot validation will not be performed on wake.

• Setting `CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON` will skip verifying the binary on every boot from power-on reset. How much time this saves depends on the binary size and the flash settings. Note that this setting carries some risk if the flash becomes corrupt unexpectedly. Read the help text of the `config item` for an explanation and recommendations if using this option.

• It’s possible to save a small amount of time during boot by disabling RTC slow clock calibration. To do so, set `CONFIG_RTC_CLK_CAL_CYCLES` to 0. Any part of the firmware that uses RTC slow clock as a timing source will be less accurate as a result.

The example project `system/startup_time` is pre-configured to optimize startup time. The file `system/startup_time/sdkconfig.defaults` contain all of these settings. You can append these to the end of your project’s own `sdkconfig` file to merge the settings, but please read the documentation for each setting first.

**Task Priorities** As ESP-IDF FreeRTOS is a real-time operating system, it’s necessary to ensure that high throughput or low latency tasks are granted a high priority in order to run immediately. Priority is set when calling `xTaskCreate()` or `xTaskCreatePinnedToCore()` and can be changed at runtime by calling `vTaskPrioritySet()`.

It’s also necessary to ensure that tasks yield CPU (by calling `vTaskDelay()`, `sleep()`, or by blocking on semaphores, queues, task notifications, etc) in order to not starve lower priority tasks and cause problems for the overall system. The Task Watchdog Timer (TWDT) provides a mechanism to automatically detect if task starvation happens, however note that a Task WDT timeout does not always indicate a problem (sometimes the correct operation of the firmware requires some long-running computation). In these cases tweaking the Task WDT timeout or even disabling the Task WDT may be necessary.

**Built-In Task Priorities** ESP-IDF starts a number of system tasks at fixed priority levels. Some are automatically started during the boot process, some are started only if the application firmware initializes a particular feature. To optimize performance, structure application task priorities so that they are not delayed by system tasks, while also not starving system tasks and impacting other functions of the system.

This may require splitting up a particular task. For example, perform a time-critical operation in a high priority task or an interrupt handler and do the non-time-critical part in a lower priority task.

Header `components/esp_system/include/esp_task.h` contains macros for the priority levels used for built-in ESP-IDF tasks system.

Common priorities are:

- **Main task that executes `app_main` function** has minimum priority (1). This task is pinned to Core 0 by default (configurable).
- **High Resolution Timer (ESP Timer)** system task to manage high precision timer events and execute callbacks has high priority (22, `ESP_TASK_TIMER_PRIO`). This task is pinned to Core 0.
- FreeRTOS Timer Task to handle FreeRTOS timer callbacks is created when the scheduler initializes and has minimum task priority (1, configurable). This task is pinned to Core 0.
- **Event Handling** system task to manage the default system event loop and execute callbacks has high priority (20, `ESP_TASK_EVENT_PRIO`) and pinned to Core 0. This configuration is only used if the application calls `esp_event_loop_create_default()`, it’s possible to call `esp_event_loop_create()` with a custom task configuration instead.
- **lwIP TCP/IP** task has high priority (18, `ESP_TASK_TCPIP_PRIO`) and is not pinned to any core (configurable).
- **Wi-Fi Driver** task has high priority (23) and is pinned to Core 0 by default (configurable).
- Wi-Fi `wpa_supplicant` component may create dedicated tasks while the Wi-Fi Protected Setup (WPS), WPA2 EAP-TLS, Device Provisioning Protocol (DPP) or BSS Transition Management (BTM) features are in use. These tasks all have low priority (2) and are not pinned to any core.
• Bluetooth Controller task has high priority (23, ESP_TASK_BT_CONTROLLER_PRIO) and is pinned to Core 0 by default (configurable). The Bluetooth Controller needs to respond to requests with low latency, so it should always be close to the highest priority task assigned to a single CPU.

• NmBLE Bluetooth Host task has high priority (21) and is pinned to Core 0 by default (configurable).

• Bluedroid Bluetooth Host creates multiple tasks when used:
  – Stack event callback task ("BTC") has high priority (19).
  – Stack BTU layer task has high priority (20).
  – Host HCI host task has high priority (22).

  All Bluedroid Tasks are pinned to the same core, which is Core 0 by default (configurable).

• The Ethernet driver creates a task for the MAC to receive Ethernet frames. If using the default config ETH_MAC_DEFAULT_CONFIG then the priority is medium-high (15) and the task is not pinned to any core. These settings can be changed by passing a custom eth_mac_config_t struct when initializing the Ethernet MAC.

• If using the MQTT component, it creates a task with default priority 5 (configurable, depends on CONFIG_MQTT_USE_CUSTOM_CONFIG) and not pinned to any core (configurable).

• To see what is the task priority for mDNS service, please check Performance Optimization.

Choosing application task priorities

With a few exceptions (most importantly the lwIP TCP/IP task), in the default configuration most built-in tasks are pinned to Core 0. This makes it quite easy for the application to place high priority tasks on Core 1. Using priority 19 or higher will guarantee an application task can run on Core 1 without being preempted by any built-in task. To further isolate the tasks running on each CPU, configure the lwIP task to only run on Core 0 instead of either core (this may reduce total TCP/IP throughput depending on what other tasks are running).

In general, it’s not recommended to set task priorities on Core 0 higher than the built-in Wi-Fi/BT operations as starving them of CPU may make the system unstable. Choosing priority 19 and Core 0 will allow lower layer Wi-Fi/BT functionality to run without delays, but still pre-empts the lwIP TCP/IP stack and other less time-critical internal functionality - this is an option for time-critical tasks that don’t perform network operations. Any task that does TCP/IP network operations should run at lower priority than the lwIP TCP/IP task (18) to avoid priority inversion issues.

备注：Setting a task to always run in preference to built-in ESP-IDF tasks does not require pinning to Core 1. The task can be left unpinned - at priority 17 or lower - to optionally run on Core 0 as well, if no higher priority built-in task is running there. Using unpinned tasks can improve the overall CPU utilization, however it makes reasoning about task scheduling more complex.

备注：Task execution is always completely suspended when writing to the built-in SPI flash chip. Only IRAM 安全中断处理程序 will continue executing.

Improving Interrupt Performance

ESP-IDF supports dynamic Interrupt allocation with interrupt preemption. Each interrupt in the system has a priority, and higher priority interrupts will preempt lower priority ones.

Interrupt handlers will execute in preference to any task (provided the task is not inside a critical section). For this reason, it’s important to minimize the amount of time spent executing in an interrupt handler.

To obtain the best performance for a particular interrupt handler:

• Assign more important interrupts a higher priority using a flag such as ESP_INTR_FLAG_LEVEL2 or ESP_INTR_FLAG_LEVEL3 when calling esp_intr_alloc().

• Assign the interrupt on a CPU where built-in Wi-Fi/BT tasks are not configured to run (this means assigning on Core 1 by default, see Built-In Task Priorities). Interrupts are assigned on the same CPU where the esp_intr_alloc() function call is made.
• If you’re sure the entire interrupt handler can run from IRAM (see IRAM 安全中断处理程序) then set the ESP_INTR_FLAG_IRAM flag when calling esp_intr_alloc() to assign the interrupt. This prevents it being temporarily disabled if the application firmware writes to the internal SPI flash.

• Even if the interrupt handler is not IRAM safe, if it is going to be executed frequently then consider moving the handler function to IRAM anyhow. This minimizes the chance of a flash cache miss when the interrupt code is executed (see Targeted Optimizations). It’s possible to do this without adding the ESP_INTR_FLAG_IRAM flag to mark the interrupt as IRAM-safe, if only part of the handler is guaranteed to be in IRAM.

### Improving Network Speed

• For Wi-Fi, see 如何提高 Wi-Fi 性能 and Wi-Fi 缓冲区使用情况

• For lwIP TCP/IP (Wi-Fi and Ethernet), see Performance Optimization

• The wifi/iperf example contains a configuration that is heavily optimized for Wi-Fi TCP/IP throughput. Append the contents of the files wifi/iperf/sdkconfig.defaults, wifi/iperf/sdkconfig.defaults.esp32 and wifi/iperf/sdkconfig.ci.99 to your project sdkconfig file in order to add all of these options. Note that some of these options may have trade-offs in terms of reduced debuggability, increased firmware size, increased memory usage, or reduced performance of other features. To get the best result, read the documentation pages linked above and use this information to determine exactly which options are best suited for your app.

### Minimizing Binary Size

The ESP-IDF build system compiles all source files in the project and ESP-IDF, but only functions and variables that are actually referenced by the program are linked into the final binary. In some cases, it is necessary to reduce the total size of the firmware binary (for example, in order to fit it into the available flash partition size).

The first step to reducing the total firmware binary size is measuring what is causing the size to increase.

#### Measuring Static Sizes

To optimize both firmware binary size and memory usage it’s necessary to measure statically allocated RAM ("data", "bss"), code ("text") and read-only data ("rodata") in your project.

Using the idf.py sub-commands size, size-components and size-files provides a summary of memory used by the project:

```
$ idf.py size
[...]
Total sizes:
  DRAM .data size: 14956 bytes
  DRAM .bss size: 15808 bytes
  Used static DRAM: 30764 bytes (149972 available, 17.0% used)
  Used static IRAM: 83918 bytes (47154 available, 64.0% used)
  Flash code: 559943 bytes
  Flash rodata: 176736 bytes
  Total image size: 835553 bytes (.bin may be padded larger)
```

This output breaks down the size of all static memory regions in the firmware binary:

• DRAM .data size is statically allocated RAM that is assigned to non-zero values at startup. This uses RAM (DRAM) at runtime and also uses space in the binary file.

• DRAM .bss size is statically allocated RAM that is assigned zero at startup. This uses RAM (DRAM) at runtime but doesn’t use any space in the binary file.
Chapter 4. API

- **Used static DRAM** is the total DRAM used by `.data + .bss`. The available size is the estimated amount of DRAM which will be available as heap memory at runtime (due to metadata overhead and implementation constraints, and heap allocations done by ESP-IDF during startup, the actual free heap at startup will be lower than this).

- **Used static IRAM** is the total size of executable code executed from IRAM. This uses space in the binary file and also reduces IRAM and/or DRAM (depending on sizes) available as heap memory at runtime. See **Optimizing IRAM Usage**.

- **Flash code** is the total size of executable code executed from flash cache (IROM). This uses space in the binary file.

- **Flash rodata** is the total size of read-only data loaded from flash cache (DROM). This uses space in the binary file.

- **Total image size** is the estimated total binary file size, which is the total of all the used memory types except for .bss.

**Component Usage Summary (idf.py size-components)** The summary output provided by idf.py size does not give enough detail to find the main contributor to excessive binary size. To analyze in more detail, use idf.py size-components

```
$ idf.py size-components
[...]
Total sizes:
DRAM .data size: 14956 bytes
DRAM .bss size: 15808 bytes
Used static DRAM: 30764 bytes (149972 available, 17.0% used)
Used static IRAM: 83918 bytes (47154 available, 64.0% used)
Flash code: 559943 bytes
Flash rodata: 176736 bytes
Total image size: 835553 bytes (.bin may be padded larger)
Per-archive contributions to ELF file:
`-`rodata Archive File DRAM .data & .bss & other IRAM D/IRAM Flash code &-
`--` 18484 Total 1267 6044 0 5490 0 107445 1
`--` 16116 liblwip.a 21 3838 0 0 0 97465 1
`--` 69907 libmbedtls.a 60 524 0 0 0 27655 1
`--` 11661 libmbedcrypto.a 64 81 0 30 0 76645 1
`--` 4708 libpp.a 2427 1292 0 20851 0 37208 1
`--` 6455 libc.a 4 0 0 0 0 57056 1
`--` 1444 libphy.a 1439 715 0 7798 0 33074 1
`--` 0 libwpa_supplicant.a 12 848 0 0 0 35505 1
`--` 1446 libfreertos.a 3104 740 0 15711 0 367 1
`--` 1913 libnvs_flash.a 0 24 0 0 0 14347 1
`--` 3817 libesp_system.a 0 4 0 0 0 5637 1
`--` 3524 libesp-tls.a 0 0 0 0 0 9165 1
[... removed some lines here ...]
`--` 0 libesp_rom.a 0 0 0 112 0 0 1
```

Espressif Systems 2331 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
Chapter 4. API

The first lines of output from `idf.py size-components` are the same as `idf.py size`. After this a table is printed of “per-archive contributions to ELF file”. This means how much each static library archive has contributed to the final binary size.

Generally, one static library archive is built per component, although some are binary libraries included by a particular component (for example, `libnet80211.a` is included by `esp_wifi` component). There are also toolchain libraries such as `libc.a` and `libgcc.a` listed here, these provide Standard C/C++ Library and toolchain built-in functionality.

If your project is simple and only has a “main” component, then all of the project’s code will be shown under `libmain.a`. If your project includes its own components (see 枚建系統), then they will each be shown on a separate line.

The table is sorted in descending order of the total contribution to the binary size.

The columns are as follows:

- **DRAM .data & .bss & other**: .data and .bss are the same as for the totals shown above (static variables, these both reduce total available RAM at runtime but .bss doesn’t contribute to the binary file size). “other” is a column for any custom section types that also contribute to RAM size (usually this value is 0).
- **IRAM**: is the same as for the totals shown above (code linked to execute from IRAM, uses space in the binary file and also reduces IRAM that can be dynamically allocated at runtime using `HEAP_CAP_32BIT`.
- **D/IRAM**: Shows IRAM space which, due to occupying D/IRAM space, is also reducing available DRAM available as heap at runtime.
- **Flash code & rodata**: these are the same as the totals above, IROM and DROM space accessed from flash cache that contribute to the binary size.

### Source File Usage Summary (idf.py size-files)

For even more detail, run `idf.py size-files` to get a summary of the contribution each object file has made to the final binary size. Each object file corresponds to a single source file.

```
$ idf.py size-files
[...]
Total sizes:
  DRAM .data size:  14956 bytes
  DRAM .bss size:  15808 bytes
  Used static DRAM:  30764 bytes ( 149972 available, 17.0% used)
  Used static IRAM:  83918 bytes ( 47154 available, 64.0% used)
  Flash code:  559943 bytes
  Flash rodata:  176736 bytes
Total image size:~ 835553 bytes (.bin may be padded larger)
Per-file contributions to ELF file:
  x509_crt_bundle.S.o  0  0  0  0  0  64212
  wl_cnx.o  2  3183  0  221  0  13119
```

(续上页)
After the summary of total sizes, a table of “Per-file contributions to ELF file” is printed.
The columns are the same as shown above for \texttt{idy.py size-components}, but this time the granularity is the contribution of each individual object file to the binary size.

For example, we can see that the file \texttt{x509.crt_bundle.S.o} contributed 64212 bytes to the total firmware size, all as \texttt{.rodata} in flash. Therefore we can guess that this application is using the \textit{ESP x509 Certificate Bundle} feature and not using this feature would save at last this many bytes from the firmware size.

Some of the object files are linked from binary libraries and therefore you won’t find a corresponding source file. To locate which component a source file belongs to, it’s generally possible to search in the ESP-IDF source tree or look in the \textit{Linker Map File} for the full path.

\textbf{Comparing Two Binaries} If making some changes that affect binary size, it’s possible to use an ESP-IDF tool to break down the exact differences in size.

This operation isn’t part of \texttt{idf.py}, it’s necessary to run the \texttt{idf_size.py} Python tool directly.

To do so, first locate the linker map file in the build directory. It will have the name \texttt{PROJECTNAME.map}. The \texttt{idf_size.py} tool performs its analysis based on the output of the linker map file.

To compare with another binary, you will also need its corresponding \texttt{.map} file saved from the build directory.

For example, to compare two builds: one with the default \texttt{CONFIG_COMPILER_OPTIMIZATION} setting “Debug (-Og)” configuration and one with “Optimize for size (-Os)”:

\begin{verbatim}
$ $IDF_PATH/tools/idf_size.py --diff build_Og/https_request.map build_Os/https_request.map
<CURRENT> MAP file: build_Os/https_request.map
<REFERENCE> MAP file: build_Og/https_request.map
Difference is counted as <CURRENT> - <REFERENCE>, i.e. a positive number means that <CURRENT> is larger.
Total sizes of <CURRENT>:

<table>
<thead>
<tr>
<th>Component</th>
<th>Size</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM .data</td>
<td>14516 bytes</td>
<td>-440 bytes</td>
</tr>
<tr>
<td>DRAM .bss</td>
<td>15792 bytes</td>
<td>-16 bytes</td>
</tr>
<tr>
<td>Used static DRAM</td>
<td>30308 bytes</td>
<td>-5420 bytes</td>
</tr>
<tr>
<td>Used static IRAM</td>
<td>78498 bytes</td>
<td>+5420 bytes</td>
</tr>
<tr>
<td>Flash code</td>
<td>509183 bytes</td>
<td>-62764 bytes</td>
</tr>
<tr>
<td>Flash rodata</td>
<td>170592 bytes</td>
<td>-6144 bytes</td>
</tr>
<tr>
<td>Total image size</td>
<td>772789 bytes</td>
<td>-62764 bytes</td>
</tr>
</tbody>
</table>

We can see from the “Difference” column that changing this one setting caused the whole binary to be over 60 KB smaller and over 5 KB more RAM is available.

It’s also possible to use the “diff” mode to output a table of component-level (static library archive) differences:

\textbf{备注：} To get the output in JSON or CSV format using \texttt{idf_size.py} it is possible to use the \texttt{--format} option.

\begin{verbatim}
$ $IDF_PATH/tools/idf_size.py --archives --diff build_Og/https_request.map build_Os/https_request.map
Also at the individual source file level:

\end{verbatim}
Other options (like writing the output to a file) are available, pass --help to see the full list.

**Showing Size When Linker Fails**  If too much static memory is used, then the linker will fail with an error such as DRAM segment data does not fit, region `iram0_0_seg' overflowed by 44 bytes, or similar.

In these cases, idf.py size will not succeed either. However it is possible to run idf_size.py manually in order to view the partial static memory usage (the memory usage will miss the variables which could not be linked, so there still appears to be some free space.)

The map file argument is `<projectname>.map` in the build directory

```
$IDF_PATH/tools/idf_size.py build/project_name.map
```

It is also possible to view the equivalent of `size-components` or `size-files` output:

```
$IDF_PATH/tools/idf_size.py --archives build/project_name.map
$IDF_PATH/tools/idf_size.py --files build/project_name.map
```

**Linker Map File**  *This is an advanced analysis method, but it can be very useful. Feel free to skip ahead to :ref:`reducing-overall-size` and possibly come back to this later.*

The idf.py size analysis tools all work by parsing the GNU binutils “linker map file”, which is a summary of everything the linker did when it created ( “linked”) the final firmware binary file

Linker map files themselves are plain text files, so it’s possible to read them and find out exactly what the linker did. However, they are also very complex and long - often 100,000 or more lines!

The map file itself is broken into parts and each part has a heading. The parts are:

- **Archive member included to satisfy reference by file (symbol).** This shows you: for each object file included in the link, what symbol (function or variable) was the linker searching for when it included that object file. If you’re wondering why some object file in particular was included in the binary, this part may give a clue. This part can be used in conjunction with the Cross Reference Table at the end of the file. Note that not every object file shown in this list ends up included in the final binary, some end up in the Discarded input sections list instead.
- **Allocating common symbols** - This is a list of (some) global variables along with their sizes. Common symbols have a particular meaning in ELF binary files, but ESP-IDF doesn’t make much use of them.
- **Discarded input sections** - These sections were read by the linker as part of an object file to be linked into the final binary, but then nothing else referred to them so they were discarded from the final binary. For ESP-IDF this list can be very long, as we compile each function and static variable to a unique section in order to minimize the final binary size (specifically ESP-IDF uses compiler options -ffunction-sections -fdata-sections and linker option --gc-sections). Items mentioned in this list do not contribute to the final binary.
- **Memory Configuration, Linker script and memory map** These two parts go together. Some of the output comes directly from the linker command line and the Linker Script, both provided by the 构建系统. The linker script is partially generated from the ESP-IDF project using the 链接器脚本生成机制 feature. As the output of the Linker script and memory map part of the map unfolds, you can see each symbol (function or static variable) linked into the final binary along with its address (as a 16 digit hex number), its length (also in hex), and the library and object file it was linked from (which can be used to determine the component and source file).
  
  Following all of the output sections that take up space in the final .bin file, the memory map also includes some sections in the ELF file that are only used for debugging (ELF sections .debug_*.*, etc.). These don’t contribute to the final binary size. You’ll notice the address of these symbols is a very low number (starting from 0x0000000000000000 and counting up).
- **Cross Reference Table.** This table shows for each symbol (function or static variable), the list of object file(s) that referred to it. If you’re wondering why a particular thing is included in the binary, this will help determine what included it.
Unfortunately, the Cross Reference Table doesn’t only include symbols that made it into the final binary. It also includes symbols in discarded sections. Therefore, just because something is shown here doesn’t mean that it was included in the final binary - this needs to be checked separately.

Linker map files are generated by the GNU binutils linker “ld”, not ESP-IDF. You can find additional information online about the linker map file format. This quick summary is written from the perspective of ESP-IDF build system in particular.

### Reducing Overall Size

The following configuration options will reduce the final binary size of almost any ESP-IDF project:

- Set `CONFIG_COMPILER_OPTIMIZATION` to “Optimize for size (-Os)” . In some cases, “Optimize for performance (-O2)” will also reduce the binary size compared to the default. Note that if your code contains C or C++ Undefined Behaviour then increasing the compiler optimization level may expose bugs that otherwise don’t happen.
- Reduce the compiled-in log output by lowering the app `CONFIG_LOG_DEFAULT_LEVEL`. If the `CONFIG_LOG_MAXIMUM_LEVEL` is changed from the default then this setting controls the binary size instead. Reducing compiled-in logging reduces the number of strings in the binary, and also the code size of the calls to logging functions.
- Set the `CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL` to “Silent”. This avoids compiling in a dedicated assertion string and source file name for each assert that may fail. It’s still possible to find the failed assert in the code by looking at the memory address where the assertion failed.
- Besides the `CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL`, you can disable or silent the assertion for HAL component separately by setting `CONFIG_HAL_DEFAULT_ASSERTION_LEVEL`. It is to notice that ESP-IDF lowers HAL assertion level in bootloader to be silent even if `CONFIG_HAL_DEFAULT_ASSERTION_LEVEL` is set to full-assertion level. This is to reduce the bootloader size.
- Set `CONFIG_COMPILER_OPTIMIZATION_CHECKS_SILENT`. This removes specific error messages for particular internal ESP-IDF error check macros. This may make it harder to debug some error conditions by reading the log output.
- If the binary needs to run on only certain revision(s) of ESP32, increasing `CONFIG_ESP32_REV_MIN` to match can result in a reduced binary size. This will make a large difference if setting ESP32 minimum revision 3, and PSRAM is enabled.
- Don’t enable `CONFIG_COMPILER_CXX.Exceptions, CONFIG_COMPILER_CXX.RTTI`, or set the `CONFIG_COMPILER_STACK_CHECK_MODE` to Overall. All of these options are already disabled by default, but they have a large impact on binary size.
- Disabling `CONFIG_ESP_ERR_TO_NAME_LOOKUP` will remove the lookup table to translate user-friendly names for error values (see 错误处理) in error logs, etc. This saves some binary size, but error values will be printed as integers only.
- Setting `CONFIG_ESP_SYSTEM_PANIC` to “Silent reboot” will save a small amount of binary size, however this is only recommended if no one will use UART output to debug the device.
- If the application binary uses only one of the security versions of the protocomm component, then the support for others can be disabled to save some code size. The support can be disabled through `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_0`, `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_1` or `CONFIG_ESP_PROTOCOMM_SUPPORT_SECURITY_VERSION_2` respectively.

In addition to the many configuration items shown here, there are a number of configuration options where changing the option from the default will increase binary size. These are not noted here. Where the increase is significant, this is usually noted in the configuration item help text.

### Targeted Optimizations

The following binary size optimizations apply to a particular component or a function:
Wi-Fi

- Disabling `CONFIG_ESP32_WIFI_ENABLE_WPA3_SAE` will save some Wi-Fi binary size if WPA3 support is not needed. (Note that WPA3 is mandatory for new Wi-Fi device certifications.)
- Disabling `CONFIG_ESP_WIFI_SOFTAP_SUPPORT` will save some Wi-Fi binary size if soft-AP support is not needed.

ADC

- Disabling ADC calibration features `CONFIG_ADC_CAL_EFUSE_TP_ENABLE`, `CONFIG_ADC_CAL_EFUSE_VREF_ENABLE`, `CONFIG_ADC_CAL_LUT_ENABLE` will save a small amount of binary size if ADC driver is used, at expense of accuracy.

Bluetooth NimBLE

If using NimBLE Bluetooth Host then the following modifications can reduce binary size:

- Set `CONFIG_BTDM_CTRL_BLE_MAX_CONN` to 1 if only one BLE connection is needed.
- `CONFIG_BT_NIMBLE_MAX_CONNECTIONS` to 1 if only one BLE connection is needed.
- Disable either `CONFIG_BT_NIMBLE_ROLE_CENTRAL` or `CONFIG_BT_NIMBLE_ROLE_OBSERVER` if these roles are not needed.
- Reducing `CONFIG_BT_NIMBLE_LOG_LEVEL` can reduce binary size. Note that if the overall log level has been reduced as described above in Reducing Overall Size then this also reduces the NimBLE log level.

lwIP IPv6

- Setting `CONFIG_LWIP_IPV6` to false will reduce the size of the lwIP TCP/IP stack, at the cost of only supporting IPv4.

备注：IPv6 is required by some components such as coap and ASIO port, These components will not be available if IPV6 is disabled.

Newlib nano formatting

By default, ESP-IDF uses newlib “full” formatting for I/O (printf, scanf, etc.)

Enabling the config option `CONFIG_NEWLIB_NANO_FORMAT` will switch newlib to the “nano” formatting mode. This both smaller in code size and a large part of the implementation is compiled into the ESP32 ROM, so it doesn’t need to be included in the binary at all.

The exact difference in binary size depends on which features the firmware uses, but 25 KB ~ 50 KB is typical.

Enabling Nano formatting also reduces the stack usage of each function that calls printf() or another string formatting function, see Reducing Stack Sizes.

“Nano” formatting doesn’t support 64-bit integers, or C99 formatting features. For a full list of restrictions, search for `--enable-newlib-nano-formatted-io` in the Newlib README file.

mbedTLS features

Under Component Config -> mbedTLS there are multiple mbedTLS features which are enabled by default but can be disabled if not needed to save code size.

These include:

- `CONFIG_MBEDTLS_HAVE_TIME`
- `CONFIG_MBEDTLS_ECDSA_DETERMINISTIC`
- `CONFIG_MBEDTLS_SHA512_C`
- `CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS`
- `CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS`
- `CONFIG_MBEDTLS_SSL_CONTEXT.Serialization`
- `CONFIG_MBEDTLS_SSL_ALPN`
- `CONFIG_MBEDTLS_SSL_RENEGOTIATION`
• CONFIG_MBEDTLS_CCM_C
• CONFIG_MBEDTLS_GCM_C
• CONFIG_MBEDTLS_ECP_C (Alternatively: Leave this option enabled but disable some of the elliptic curves listed in the sub-menu.)
• Change CONFIG_MBEDTLS_TLS_MODE if both server & client functionalities are not needed
• Consider disabling some ciphersuites listed in the “TLS Key Exchange Methods” sub-menu (i.e. CONFIG_MBEDTLS_KEY_EXCHANGE_RSA)

The help text for each option has some more information.

**Important:** It is strongly not recommended to disable all these mbedTLS options. Only disable options where you understand the functionality and are certain that it is not needed in the application. In particular:

- Ensure that any TLS server(s) the device connects to can still be used. If the server is controlled by a third party or a cloud service, recommend ensuring that the firmware supports at least two of the supported cipher suites in case one is disabled in a future update.
- Ensure that any TLS client(s) that connect to the device can still connect with supported/recommended cipher suites. Note that future versions of client operating systems may remove support for some features, so it is recommended to enable multiple supported cipher suites or algorithms for redundancy.

If depending on third party clients or servers, always pay attention to announcements about future changes to supported TLS features. If not, the ESP32 device may become inaccessible if support changes.

**Note:** Not every combination of mbedTLS compile-time config is tested in ESP-IDF. If you find a combination that fails to compile or function as expected, please report the details on GitHub.

**VFS** Virtual filesystem feature in ESP-IDF allows multiple filesystem drivers and file-like peripheral drivers to be accessed using standard I/O functions (`open`, `read`, `write`, etc.) and C library functions (`fopen`, `fread`, `fwrite`, etc.). When filesystem or file-like peripheral driver functionality is not used in the application this feature can be fully or partially disabled. VFS component provides the following configuration options:

- **CONFIG_VFS_SUPPORT_TERMIOS** — can be disabled if the application doesn’t use `termios` family of functions. Currently, these functions are implemented only for UART VFS driver. Most applications can disable this option. Disabling this option reduces the code size by about 1.8 kB.
- **CONFIG_VFS_SUPPORT_SELECT** — can be disabled if the application doesn’t use `select` function with file descriptors. Currently, only the UART and eventfd VFS drivers implement `select` support. Note that when this option is disabled, `select` can still be used for socket file descriptors. Disabling this option reduces the code size by about 2.7 kB.
- **CONFIG_VFS_SUPPORT_DIR** — can be disabled if the application doesn’t use directory related functions, such as `readdir` (see the description of this option for the complete list). Applications which only open, read and write specific files and don’t need to enumerate or create directories can disable this option, reducing the code size by 0.5 kB or more, depending on the filesystem drivers in use.
- **CONFIG_VFS_SUPPORT_IO** — can be disabled if the application doesn’t use filesystems or file-like peripheral drivers. This disables all VFS functionality, including the three options mentioned above. When this option is disabled, `console` can’t be used. Note that the application can still use standard I/O functions with socket file descriptors when this option is disabled. Compared to the default configuration, disabling this option reduces code size by about 9.4 kB.

**Bootloader Size** This document deals with the size of an ESP-IDF app binary only, and not the ESP-IDF 二级引导程序.

For a discussion of ESP-IDF bootloader binary size, see 引导加载程序大小.

**IRAM Binary Size** If the IRAM section of a binary is too large, this issue can be resolved by reducing IRAM memory usage. See Optimizing IRAM Usage.
Minimizing RAM Usage

In some cases, a firmware application’s available RAM may run low or run out entirely. In these cases, it’s necessary to tune the memory usage of the firmware application.

In general, firmware should aim to leave some “headroom” of free internal RAM in order to deal with extraordinary situations or changes in RAM usage in future updates.

Background  Before optimizing ESP-IDF RAM usage, it’s necessary to understand the basics of ESP32 memory types, the difference between static and dynamic memory usage in C, and the way ESP-IDF uses stack and heap. This information can all be found in Heap Memory Allocation.

Measuring Static Memory Usage  The idf.py tool can be used to generate reports about the static memory usage of an application. Refer to the Binary Size chapter for more information.

Measuring Dynamic Memory Usage  ESP-IDF contains a range of heap APIs for measuring free heap at runtime. See Heap Memory Debugging.

备注： In embedded systems, heap fragmentation can be a significant issue alongside total RAM usage. The heap measurement APIs provide ways to measure the “largest free block”. Monitoring this value along with the total number of free bytes can give a quick indication of whether heap fragmentation is becoming an issue.

Reducing Static Memory Usage

• Reducing the static memory usage of the application increases the amount of RAM available for heap at runtime, and vice versa.
• Generally speaking, minimizing static memory usage requires monitoring the .data and .bss sizes. For tools to do this, see Measuring Static Sizes.
• Internal ESP-IDF functions do not make heavy use of static RAM allocation in C. In many instances (including: Wi-Fi library, Bluetooth controller) “static” buffers are still allocated from heap, but the allocation is done once when the feature is initialized and will be freed if the feature is deinitialized. This is done in order to maximize the amount of free memory at different points in the application life-cycle.

To minimize static memory use:

• Declare structures, buffers, or other variables const whenever possible. Constant data can be stored in flash not RAM. This may require changing functions in the firmware to take const * arguments instead of mutable pointer arguments. These changes can also reduce the stack usage of some functions.
• If using Bluedroid, setting the option CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY will cause Bluedroid to allocate memory on initialization and free it on deinitialization. This doesn’t necessarily reduce the peak memory usage, but changes it from static memory usage to runtime memory usage.

Reducing Stack Sizes  In FreeRTOS, task stacks are usually allocated from the heap. The stack size for each task is fixed (passed as an argument to xTaskCreate()). Each task can use up to its allocated stack size, but using more than this will cause an otherwise valid program to crash with a stack overflow or heap corruption.

Therefore, determining the optimum sizes of each task stack can substantially reduce RAM usage.

To determine optimum task stack sizes:

• Combine tasks. The best task stack size is 0 bytes, achieved by combining a task with another existing task. Anywhere that the firmware can be structured to perform multiple functions sequentially in a single task will increase free memory. In some cases, using a “worker task” pattern where jobs are serialized into a FreeRTOS queue (or similar) and then processed by generic worker tasks may help.
• Consolidate task functions. String formatting functions (like printf) are particularly heavy users of stack, so any task which doesn’t ever call these can usually have its stack size reduced.

• Enabling Newlib nano formatting will reduce the stack usage of any task that calls printf() or other C string formatting functions.

• Avoid allocating large variables on the stack. In C, any large struct or array allocated as an “automatic” variable (i.e. default scope of a C declaration) will use space on the stack. Minimize the sizes of these, allocate them statically and/or see if you can save memory by allocating them from the heap only when they are needed.

• Avoid deep recursive function calls. Individual recursive function calls don’t always add a lot of stack usage each time they are called, but if each function includes large stack-based variables then the overhead can get quite high.

• At runtime, call the function uxTaskGetStackHighWaterMark() with the handle of any task where you think there is unused stack memory. This function returns the minimum lifetime free stack memory in bytes. The easiest time to call this is from the task itself: call uxTaskGetStackHighWaterMark(NULL) to get the current task’s high water mark after the time that the task has achieved its peak stack usage (i.e. if there is a main loop, execute the main loop a number of times with all possible states and then call uxTaskGetStackHighWaterMark()). Often, it’s possible to subtract almost the entire value returned here from the total stack size of a task, but allow some safety margin to account for unexpected small increases in stack usage at runtime.

• Call uxTaskGetSystemState() at runtime to get a summary of all tasks in the system. This includes their individual stack “high watermark” values.

• When debugger watchpoints are not being used, set the CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK option to trigger an immediate panic if a task writes the word at the end of its assigned stack. This is slightly more reliable than the default CONFIG_FREERTOS_CHECK_STACKOVERFLOW option of “Check using canary bytes”, because the panic happens immediately, not on the next RTOS context switch. Neither option is perfect, it’s possible in some cases for stack pointer to skip the watchpoint or canary bytes and corrupt another region of RAM, instead.

**Internal Stack Sizes**

ESP-IDF allocates a number of internal tasks for housekeeping purposes or operating system functions. Some are created during the startup process, and some are created at runtime when particular features are initialized.

The default stack sizes for these tasks are usually set conservatively high, to allow all common usage patterns. Many of the stack sizes are configurable, and it may be possible to reduce them to match the real runtime stack usage of the task.

重 要：If internal task stack sizes are set too small, ESP-IDF will crash unpredictably. Even if the root cause is task stack overflow, this is not always clear when debugging. It is recommended that internal stack sizes are only reduced carefully (if at all), with close attention to “high water mark” free space under load. If reporting an issue that occurs when internal task stack sizes have been reduced, please always include this information and the specific configuration that is being used.

• **Main task that executes app_main function** has stack size CONFIG_ESP_MAIN_TASK_STACK_SIZE.

• **High Resolution Timer (ESP Timer) system task which executes callbacks** has stack size CONFIG_ESP_TIMER_TASK_STACK_SIZE.

• FreeRTOS Timer Task to handle FreeRTOS timer callbacks has stack size CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH.

• **Event Handling** system task to execute callbacks for the default system event loop has stack size CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE.

• lwIP TCP/IP task has stack size CONFIG_LWIP_TCP_IP_TASK_STACK_SIZE.

• BlueDroid Bluetooth Host have task stack sizes CONFIG_BT_BTC_TASK_STACK_SIZE, CONFIG_BT_BTU_TASK_STACK_SIZE.

• NimBLE Bluetooth Host has task stack size CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE.

• The Ethernet driver creates a task for the MAC to receive Ethernet frames. If using the default config ETH_MAC_DEFAULT_CONFIG then the task stack size is 4 KB. This setting can be changed by passing a custom eth_mac_config_t struct when initializing the Ethernet MAC.
Chapter 4. API

- FreeRTOS idle task stack size is configured by `CONFIG_FREERTOS_IDLE_TASK_STACKSIZE`.
- If using the MQTT component, it creates a task with stack size configured by `CONFIG_MQTT_TASK_STACK_SIZE`. MQTT stack size can also be configured using `task_stack` field of `esp_mqtt_client_config_t`.
- To see how to optimize RAM usage when using mDNS, please check Performance Optimization.

备注：Aside from built-in system features such as esp-timer, if an ESP-IDF feature is not initialized by the firmware then no associated task is created. In those cases, the stack usage is zero and the stack size configuration for the task is not relevant.

Reducing Heap Usage  For functions that assist in analyzing heap usage at runtime, see Heap Memory Debugging.

Normally, optimizing heap usage consists of analyzing the usage and removing calls to `malloc()` that aren’t being used, reducing the corresponding sizes, or freeing previously allocated buffers earlier.

There are some ESP-IDF configuration options that can reduce heap usage at runtime:

- lwIP documentation has a section to configure Minimum RAM usage.
- Wi-Fi缓冲区使用情况 describes options to either reduce numbers of “static” buffers or reduce the maximum number of “dynamic” buffers in use, in order to minimize memory usage at possible cost of performance. Note that “static” Wi-Fi buffers are still allocated from heap when Wi-Fi is initialized and will be freed if Wi-Fi is deinitialized.
- The Ethernet driver allocates DMA buffers for the internal Ethernet MAC when it is initialized - configuration options are `CONFIG_ETH_DMA_BUFFER_SIZE`, `CONFIG_ETH_DMA_RX_BUFFER_NUM`, `CONFIG_ETH_DMA_TX_BUFFER_NUM`.
- Several Mbed TLS configuration options can be used to reduce heap memory usage. See the Mbed TLS docs for details.
- In single core mode only, it’s possible to use IRAM as byte accessible memory (added to the regular heap) by enabling `CONFIG_ESP32_IRAM_AS_8BIT_ACCESSIBLE_MEMORY`. Note that this option carries a performance penalty and the risk of security issues caused by executable data. If this option is enabled then it’s possible to set other options to prefer certain buffers be allocated from this memory: mbedTLS, NimBLE.
- Reduce `CONFIG_BTDM_CTRL_BLE_MAX_CONN` if using BLE.
- Reduce `CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN` if using Bluetooth Classic.

备注：There are other configuration options that will increase heap usage at runtime if changed from the defaults. These are not listed here, but the help text for the configuration item will mention if there is some memory impact.

Optimizing IRAM Usage  If the app allocates more static IRAM than is available then the app will fail to build and linker errors such as section `.iram0.text` will not fit in region `.iram0_0_seg`. IRAM0 segment data does not fit and region `.iram0_0_seg` overflowed by 84 bytes will be seen. If this happens, it is necessary to find ways to reduce static IRAM usage in order to link the application.

To analyze the IRAM usage in the firmware binary, use Measuring Static Sizes. If the firmware failed to link, steps to analyze are shown at Showing Size When Linker Fails.

The following options will reduce IRAM usage of some ESP-IDF features:

- Enable `CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH`. Provided these functions are not (incorrectly) used from ISRs, this option is safe to enable in all configurations.
- Enable `CONFIG_FREERTOSPLACE_SNAPSHOT_FUNS_INTO_FLASH`. Enabling this option will place snapshot-related functions, such as vTaskGetSnapshot or uxTaskGetSnapshotAll, in flash.
- Enable `CONFIG_RINGBUFPPLACE_FUNCTIONS_INTO_FLASH`. Provided these functions are not (incorrectly) used from ISRs, this option is safe to enable in all configurations.
Chapter 4. API 指南

- Enable `CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH`. This option is not safe to use if the ISR ringbuf functions are used from an IRAM interrupt context, e.g. if `CONFIG_UART_ISR_IN_IRAM` is enabled. For the IDF drivers where this is the case you will get an error at run-time when installing the driver in question.
- Disable Wi-Fi options `CONFIG_ESP32_WIFI_IRAM_OPT` and/or `CONFIG_ESP32_WIFI_RX_IRAM_OPT`. Disabling these options will free available IRAM at the cost of Wi-Fi performance.
- `CONFIG_SPI_FLASH_ROM_DRIVER_PATCH` disabling this option will free some IRAM but is only available in some flash configurations (see the configuration item help text).
- If the application uses PSRAM and is based on ESP32 rev. 3 (ECO3), setting `CONFIG_ESP32_REV_MIN` to 3 will disable PSRAM bug workarounds, saving ~10kB or more of IRAM.
- Disabling `CONFIG_ESP32_WIFI_IRAM_OPT` prevents posting `esp_event` events from IRAM 安全中断处理程序 but will save some IRAM.
- Disabling `CONFIG_SPI_MASTER_ISR_IN_IRAM` prevents `spi_master` interrupts from being serviced while writing to flash, and may otherwise reduce `spi_master` performance, but will save some IRAM.
- Setting `CONFIG_HAL_DEFAULT_ASSERTION_LEVEL` to disable assertion for HAL component will save some IRAM especially for HAL code who calls `HAL_ASSERT` a lot and resides in IRAM.

Putting C Library in Flash When compiling for ESP32 revisions older than ECO3 (`CONFIG_ESP32_REV_MIN`), PSRAM cache bug workaround (`CONFIG_SPIRAM_CACHE_WORKAROUND`) option is enabled, and the C library functions normally located in ROM are recompiled with the workaround and placed into IRAM instead. For most applications, it is safe to move many of the C library functions into Flash, claiming some IRAM. Corresponding options include:

- `CONFIG_SPIRAM_CACHE_LIBIMP_IN_IRAM`: affects the functions `longjmp` and `setjmp`.
- `CONFIG_SPIRAM_CACHE_LIBMATH_IN_IRAM`: affects the functions `abs`, `div`, `labs`, `ldiv`, `quorem`, `fpclassify` and `nan`.
- `CONFIG_SPIRAM_CACHE_LIBNUMPARSER_IN_IRAM`: affects the functions `utoa`, `itoa`, `atoi`, ` atol`, `strtol`, and `strtoul`.
- `CONFIG_SPIRAM_CACHE_LIBBIO_IN_IRAM`: affects the functions `wctomb`, `fwrite`, `wbuf`, `wsetup`, `fpwrite`, `wctomb_r`, `ungetc`, `makebuf`, `fllush`, `refill`, and `sccl`.
- `CONFIG_SPIRAM_CACHE_LIBTIME_IN_IRAM`: affects the functions `asctime`, `asctime_r`, `ctime`, `ctime_r`, `lcltime`, `lcltime_r`, `gmtime`, `gmtime_r`, `strftime`, `mktime`, `tzset`, `time`, `gettinfo`, `system`, `month_lengths`, `timelocal`, `tzvars`, `tzlock`, `tzcalc_limits`, and `strptime`.
- `CONFIG_SPIRAM_CACHE_LIBCHAR_IN_IRAM`: affects the functions `ctype_r`, `toupper`, `tolower`, `toascii`, `strupr`, `bzero`, `isalnum`, `isalpha`, `isascii`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `islower`, `ispunct`, `isspace`, and `isupper`.
- `CONFIG_SPIRAM_CACHE_LIBMEM_IN_IRAM`: affects the functions `memcpy`, `memchr`, `memmove`, and `memchr`.
- `CONFIG_SPIRAM_CACHE Libstrr_IN_IRAM`: affects the functions `stristr`, `strncmp`, `strndup`, `strdup`, `strndup_r`, `strftime`, `strleq`, `strleqr`, `strchr`, `strcoll`, `strncpy`, `strncpy_r`, `strlen`, `strlwr`, `strncasecmp`, `strncat`, `strncpy`, `strndup`, `strndup_r`, `strchr`, `strsep`, `strspn`, `strstr`, `strtok`, and `strupr`.
- `CONFIG_SPIRAM_CACHE_LIBRAND_IN_IRAM`: affects the functions `rand`, `rand`, and `rand_r`.
- `CONFIG_SPIRAM_CACHE_LIBENV_IN_IRAM`: affects the functions `environ`, `environlock`, and `getenv_r`.
- `CONFIG_SPIRAM_CACHE_LIBFILE_IN_IRAM`: affects the functions `lock_t`, `isatty`, `fclose`, `open`, `close`, `creat`, ` creat`, `read`, `rshift`, `sbrk`, `st dio`, `syssbrk`, `sysclose`, `sysopen`, `creat`, `sysread`, `syswrite`, `impure`, `fwrite`, and `findfp`.
- `CONFIG_SPIRAM_CACHE_LIBMISC_IN_IRAM`: affects the functions `raise` and `system`.

The exact amount of IRAM saved will depend on how much C library code is actually used by the application. In addition to these, the following options may be used to move more of the C library code into Flash, however note that this may result in reduced performance. Also take care to not use corresponding C library functions from interrupts which may be called while cache is disabled (allocated with `ESP_INTR_FLAG_IRAM` flag), refer to IRAM 安全中断处理程序 for more details. For these reasons, the functions `itoa`, `memcmp`, `memcpy`, `memset`, `strcat`, `strcp y`, and `strlen` are always put in IRAM.
Moving frequently-called functions from IRAM to flash may increase their execution time.

Other configuration options exist that will increase IRAM usage by moving some functionality into IRAM, usually for performance, but the default option is not to do this. These are not listed here. The IRAM size impact of enabling these options is usually noted in the configuration item help text.

### 4.24 RF calibration

ESP32 supports three RF calibration methods during RF initialization:

1. Partial calibration
2. Full calibration
3. No calibration

#### 4.24.1 Partial calibration

During RF initialization, the partial calibration method is used by default for RF calibration. It is done based on the full calibration data which is stored in the NVS. To use this method, please go to `menuconfig` and enable `CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE`.

#### 4.24.2 Full calibration

Full calibration is triggered in the following conditions:

1. NVS does not exist.
2. The NVS partition to store calibration data is erased.
3. Hardware MAC address is changed.
4. PHY library version is changed.
5. The RF calibration data loaded from the NVS partition is broken.

It takes about 100ms more than partial calibration. If boot duration is not critical, it is suggested to use the full calibration method. To switch to the full calibration method, go to `menuconfig` and disable `CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE`. If you use the default method of RF calibration, there are two ways to add the function of triggering full calibration as a last-resort remedy.

1. Erase the NVS partition if you don’t mind all of the data stored in the NVS partition is erased. That is indeed the easiest way.
2. Call API `esp_phy_erase_cal_data_in_nvs()` before initializing WiFi and BT/BLE based on some conditions (e.g., an option provided in some diagnostic mode). In this case, only phy namespace of the NVS partition is erased.

#### 4.24.3 No calibration

No calibration method is only used when the device wakes up from deep sleep.

#### 4.24.4 PHY initialization data

The PHY initialization data is used for RF calibration. There are two ways to get the PHY initialization data.

One is the default initialization data which is located in the header file `components/esp_phy/esp32/include/phy_init_data.h`. 
It is embedded into the application binary after compiling and then stored into read-only memory (DROM). To use the default initialization data, please go to menuconfig and disable `CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION`.

Another is the initialization data which is stored in a partition. When using a custom partition table, make sure that PHY data partition is included (type: `data`, subtype: `phy`). With default partition table, this is done automatically. If initialization data is stored in a partition, it has to be flashed there, otherwise runtime error will occur. To switch to the initialization data stored in a partition, go to menuconfig and enable `CONFIG_ESP_PHY_INIT_DATA_IN_PARTITION`.

### 4.24.5 API Reference

#### Header File

- `components/esp_phy/include/esp_phy_init.h`

#### Functions

**const esp_phy_init_data_t** `*esp_phy_get_init_data (void)`

Get PHY init data.

If “Use a partition to store PHY init data” option is set in menuconfig, this function will load PHY init data from a partition. Otherwise, PHY init data will be compiled into the application itself, and this function will return a pointer to PHY init data located in read-only memory (DROM).

If “Use a partition to store PHY init data” option is enabled, this function may return NULL if the data loaded from flash is not valid.

备注: Call `esp_phy_release_init_data` to release the pointer obtained using this function after the call to `esp_wifi_init`.

返回 pointer to PHY init data structure

**void** `esp_phy_release_init_data (const esp_phy_init_data_t *data)`

Release PHY init data.

参数 data - pointer to PHY init data structure obtained from `esp_phy_get_init_data` function

**esp_err_t** `esp_phy_load_cal_data_from_nvs (esp_phy_calibration_data_t *out_cal_data)`

Function called by `esp_phy_init` to load PHY calibration data.

This is a convenience function which can be used to load PHY calibration data from NVS. Data can be stored to NVS using `esp_phy_store_cal_data_to_nvs` function.

If calibration data is not present in the NVS, or data is not valid (was obtained for a chip with a different MAC address, or obtained for a different version of software), this function will return an error.

If “Initialize PHY in startup code” option is set in menuconfig, this function will be used to load calibration data. To provide a different mechanism for loading calibration data, disable “Initialize PHY in startup code” option in menuconfig and call `esp_phy_init` function from the application. For an example usage of `esp_phy_init` and this function, see `esp_phy_store_cal_data_to_nvs` function in `cpu_start.c`.

参数 out_cal_data - pointer to calibration data structure to be filled with loaded data.

返回 ESP_OK on success

**esp_err_t** `esp_phy_store_cal_data_to_nvs (const esp_phy_calibration_data_t *cal_data)`

Function called by `esp_phy_init` to store PHY calibration data.

This is a convenience function which can be used to store PHY calibration data to the NVS. Calibration data is returned by `esp_phy_init` function. Data saved using this function to the NVS can later be loaded using `esp_phy_store_cal_data_to_nvs` function.
If “Initialize PHY in startup code” option is set in menuconfig, this function will be used to store calibration
data. To provide a different mechanism for storing calibration data, disable “Initialize PHY in startup code”
option in menuconfig and call esp_phy_init function from the application.

参数 cal_data - pointer to calibration data which has to be saved.

返回 ESP_OK on success

**esp_err_t esp_phy_erase_cal_data_in_nvs (void)**

Erase PHY calibration data which is stored in the NVS.

This is a function which can be used to trigger full calibration as a last-resort remedy if partial calibration is
used. It can be called in the application based on some conditions (e.g. an option provided in some diagnostic
mode).

返回 ESP_OK on success

返回 others on fail. Please refer to NVS API return value error number.

void esp_phy_enable (void)

Enable PHY and RF module.

PHY and RF module should be enabled in order to use WiFi or BT. Now PHY and RF enabling job is done
automatically when start WiFi or BT. Users should not call this API in their application.

void esp_phy_disable (void)

Disable PHY and RF module.

PHY module should be disabled in order to shutdown WiFi or BT. Now PHY and RF disabling job is done
automatically when stop WiFi or BT. Users should not call this API in their application.

void esp_phy_load_cal_and_init (void)

Load calibration data from NVS and initialize PHY and RF module.

void esp_phy_pd_mem_init (void)

Initialize backup memory for Phy power up/down.

void esp_phy_pd_mem_deinit (void)

Deinitialize backup memory for Phy power up/down.

void esp_phy_common_clock_enable (void)

Enable WiFi/BT common clock.

void esp_phy_common_clock_disable (void)

Disable WiFi/BT common clock.

int64_t esp_phy_rf_get_on_ts (void)

Get the timestamp when PHY/RF was switched on.

返回 return 0 if PHY/RF is never switched on. Otherwise return time in microsecond since boot
when phy/rf was last switched on

**esp_err_t esp_phy_update_country_info (const char *country)**

Update the corresponding PHY init type according to the country code of Wi-Fi.

参数 country – country code

返回 ESP_OK on success.

返回 esp_err_t code describing the error on fail

char *get_phy_version_str (void)

Get PHY lib version.

返回 PHY lib version.
Structures

struct esp_phy_init_data_t
  Structure holding PHY init parameters.

  **Public Members**

  uint8_t params[128]
  opaque PHY initialization parameters

struct esp_phy_calibration_data_t
  Opaque PHY calibration data.

  **Public Members**

  uint8_t version[4]
  PHY version

  uint8_t mac[6]
  The MAC address of the station

  uint8_t opaque[1894]
  calibration data

Enumerations

enum esp_phy_calibration_mode_t
  PHY calibration mode.

  **Values:**

  enumerator PHY_RF_CAL_PARTIAL
  Do part of RF calibration. This should be used after power-on reset.

  enumerator PHY_RF_CAL_NONE
  Don’t do any RF calibration. This mode is only suggested to be used after deep sleep reset.

  enumerator PHY_RF_CAL_FULL
  Do full RF calibration. Produces best results, but also consumes a lot of time and current. Suggested to be used once.

4.25 Security

This guide provides an overview of the overall security features available in Espressif solutions. It is highly recommended to consider this guide while designing the products with Espressif platform and ESP-IDF software stack from the “security” perspective.
4.25.1 Goals

High level security goals are as follows:

1. Preventing untrusted code execution
2. Protecting the identity and integrity of the code stored in the off-chip flash memory
3. Securing device identity
4. Secure storage for confidential data
5. Authenticated and encrypted communication from the device

4.25.2 Platform Security

Secure Boot

Secure Boot feature ensures that only authenticated software can execute on the device. Secure boot process forms chain of trust by verifying all mutable software entities involved in the ESP-IDF boot process. Signature verification happens during both boot-up as well as OTA updates.

Please refer to the Secure Boot (v2) Guide for detailed documentation about this feature.

For ESP32 before ECO3, please refer to Secure Boot (v1) Guide.

It is highly recommended that a secure boot feature be enabled on all production devices.

Secure Boot Best Practices

• Generate the signing key on a system with a quality source of entropy.
• Always keep the signing key private. A leak of this key will compromise the Secure Boot system.
• Do not allow any third party to observe any aspects of the key generation or signing process using espsecure.py. Both processes are vulnerable to timing or other side-channel attacks.
• Ensure that all security eFuses have been correctly programmed, includes disabling of the debug interfaces, non-required boot mediums (e.g., UART DL mode) etc.

Flash Encryption

Flash Encryption feature helps to encrypt the contents on the off-chip flash memory and thus provides the confidentiality aspect to the software or data stored in the flash memory.

Please refer to the Flash Encryption Guide for detailed documentation about this feature.

Flash Encryption Best Practices

• It is recommended to use Flash Encryption release mode for the production use-cases.
• It is recommended to have a unique flash encryption key per device.
• Enable Secure Boot as an extra layer of protection, and to prevent an attacker from selectively corrupting any part of the flash before boot.

Debug Interfaces

JTAG

• JTAG interfaces stays disabled if any of the security features are enabled, please refer to JTAG 与闪存加密和安全引导 for more information.
• JTAG interface can also be disabled in the absence of any other security features using eFuse API.
Chapter 4. API 指南

UART DL Mode  For ESP32 ECO3 case, UART Download mode stays disabled if any of the security features are enabled in their release configuration. Alternatively, it can also be disabled by calling `esp_efuse_disable_rom_download_mode()` at runtime.

重要：If UART Download mode is disabled then `esptool` cannot work on the device.

4.25.3  Network Security

Wi-Fi

In addition to the traditional security methods (WEP/WPA-TKIP/WPA2-CCMP), Wi-Fi driver in ESP-IDF also supports additional state-of-the-art security protocols. Please refer to the Wi-Fi Security for detailed documentation.

TLS (Transport Layer Security)

It is recommended to use TLS (Transport Layer Security) in all external communications, e.g., cloud communication, OTA updates etc. from the ESP device. ESP-IDF supports mbedTLS as the official TLS stack.

TLS is default integrated in ESP HTTP Client, ESP HTTPS Server and several other components that ship with ESP-IDF.

备注：It is recommended to use ESP-IDF protocol components in their default configuration which has been ensured to be secure. Disabling HTTPS and similar security critical configurations should be avoided.

ESP-TLS Abstraction  ESP-IDF provides an abstraction layer for most used TLS functionalities and hence it is recommended that an application makes use of the API exposed by ESP-TLS.

TLS Server verification section highlights diverse ways in which the identity of server could be established on the device side.

ESP Certificate Bundle  The ESP x509 Certificate Bundle API provides an easy way to include a bundle of custom x509 root certificates for TLS server verification. The certificate bundle is the easiest way to verify the identity of almost all standard TLS servers.

重要：It is highly recommended to verify the identity of the server (based on X.509 certificates) to avoid establishing communication with the fake server.

4.25.4  Product Security

Secure Provisioning

Secure Provisioning refers to a process of secure on-boarding of the ESP device on to the Wi-Fi network. This mechanism also allows provision of additional custom configuration data during the initial provisioning phase from the provisioning entity (e.g., Smartphone).

ESP-IDF provides various security schemes to establish a secure session between ESP and the provisioning entity, they are highlighted at Security Schemes.

Please refer to the Wi-Fi Provisioning documentation for details and example code for this feature.
Secure OTA (Over-the-air) Updates

- OTA Updates must happen over secure transport, e.g., HTTPS.
- ESP-IDF provides a simplified abstraction layer ESP HTTPS OTA for this.
- If Secure Boot is enabled then server should host the signed application image.
- If Flash Encryption is enabled then no additional steps are required on the server side, encryption shall be taken care on the device itself during flash write.
- OTA update 回滚过程 can help to switch the application as active only after its functionality has been verified.

Anti-Rollback Protection  Anti-roll back protection feature ensures that device only executes application that meets the security version criteria as stored in its eFuse. So even though the application is trusted and signed by legitimate key it may contain some revoked security feature or credential and hence device must reject any such application.

ESP-IDF allows this feature for the application only and it’s managed through 2nd stage bootloader. The security version is stored in the device eFuse and it’s compared against the application image header during both bootup and over-the-air updates.

Please see more information to enable this feature in the 回滚 guide.

Encrypted Firmware Distribution  Encrypted firmware distribution during over-the-air updates ensure that the application stays encrypted in transit from server to the device. This can act as an additional layer of protection on top of the TLS communication during OTA updates and protect the identity of the application.

Please see working example for this documented in OTA Upgrades with Pre-Encrypted Firmware section.

Secure Storage

Secure storage refers to the application specific data that can be stored in a secure manner on the device (off-chip flash memory). This is typically read-write flash partition and holds device specific configuration data e.g., Wi-Fi credentials.

ESP-IDF provides “NVS (Non-volatile Storage)” management component which allows encrypted data partitions. This feature is tied with the platform Flash Encryption feature described earlier.

Please refer to the NVS Encryption for detailed documentation on the working and instructions to enable this feature.

Secure Device Control

ESP-IDF provides capability to control an ESP device over Wi-Fi + HTTP or BLE in a secure manner using ESP Local Control component.

Please refer to the ESP Local Control for detailed documentation about this feature.

4.25.5 Security Policy

ESP-IDF GitHub repository has attached Security Policy Brief.
Adviseories

- Espressif publishes critical Security Advisories on the website, this includes both hardware and software related.
- ESP-IDF software components specific advisories are published through the GitHub repository.

Software Updates

Critical security issues in the ESP-IDF components, 3rd party libraries are fixed as and when we find them or when they are reported to us. Gradually, we make the fixes available in all applicable release branches in ESP-IDF.

Applicable security issues and CVEs for the ESP-IDF components, 3rd party libraries are mentioned in the ESP-IDF release notes.

We recommend periodically updating to the latest bugfix version of the ESP-IDF release to have all critical security fixes available.

4.26 Secure Boot

All references in this document are related to Secure Boot V1 (The AES based Secure Boot Scheme). ESP32 Revision 3 onwards, the preferred secure boot scheme is Secure Boot V2. Please refer to Secure Boot V2 document for ESP32 Revision 3 or ESP32-S2.

Secure Boot is a feature for ensuring only your code can run on the chip. Data loaded from flash is verified on each reset.

Secure Boot is separate from the Flash Encryption feature, and you can use secure boot without encrypting the flash contents. However, for a secure environment both should be used simultaneously. See Secure Boot & Flash Encryption for more details.

Enabling secure boot limits your options for further updates of your ESP32. Make sure to read this document throughly and understand the implications of enabling secure boot.

4.26.1 Background

- Most data is stored in flash. Flash access does not need to be protected from physical access in order for secure boot to function, because critical data is stored (non-software-accessible) in Efuses internal to the chip.
- Efuses are used to store the secure bootloader key (in efuse BLOCK2), and also a single Efuse bit (ABS_DONE_0) is burned (written to 1) to permanently enable secure boot on the chip. For more details on Efuses, see ESP32 Technical Reference Manual > eFuse Controller (eFuse) [PDF].
- To understand the secure boot process, first familiarise yourself with the standard ESP-IDF boot process.
- Both stages of the boot process (initial software bootloader load, and subsequent partition & app loading) are verified by the secure boot process, in a “chain of trust” relationship.

4.26.2 Secure Boot Process Overview

This is a high level overview of the secure boot process. Step by step instructions are supplied under How To Enable Secure Boot. Further in-depth details are supplied under Technical Details:

1. The options to enable secure boot are provided in the Project Configuration Menu, under “Secure Boot Configuration”.
2. Secure Boot defaults to signing images and partition table data during the build process. The “Secure boot private signing key” config item is a file path to a ECDSA public/private key pair in a PEM format file.
3. The software bootloader image is built by esp-idf with secure boot support enabled and the public key (signature verification) portion of the secure boot signing key compiled in. This software bootloader image is flashed at offset 0x1000.
4. On first boot, the software bootloader follows the following process to enable secure boot:
   - Hardware secure boot support generates a device secure bootloader key (generated via hardware RNG, then stored read/write protected in efuse), and a secure digest. The digest is derived from the key, an IV, and the bootloader image contents.
   - The secure digest is flashed at offset 0x0 in the flash.
   - Depending on Secure Boot Configuration, efuses are burned to disable JTAG and the ROM BASIC interpreter (it is strongly recommended these options are turned on.)
   - Bootloader permanently enables secure boot by burning the ABS_DONE_0 efuse. The software bootloader then becomes protected (the chip will only boot a bootloader image if the digest matches.)
5. On subsequent boots the ROM bootloader sees that the secure boot efuse is burned, reads the saved digest at 0x0 and uses hardware secure boot support to compare it with a newly calculated digest. If the digest does not match then booting will not continue. The digest and comparison are performed entirely by hardware, and the calculated digest is not readable by software. For technical details see Secure Boot Hardware Support.

4.26.3 Keys

The following keys are used by the secure boot process:

- “secure bootloader key” is a 256-bit AES key that is stored in Efuse block 2. The bootloader can generate this key itself from the internal hardware random number generator, the user does not need to supply it (it is optionally possible to supply this key, see Re-Flashable Software Bootloader). The Efuse holding this key is read & write protected (preventing software access) before secure boot is enabled.
- By default, the Efuse Block 2 Coding Scheme is “None” and a 256 bit key is stored in this block. On some ESP32s, the Coding Scheme is set to 3/4 Encoding (CODING_SCHEME efuse has value 1) and a 192 bit key must be stored in this block.

For more details, see ESP32 Technical Reference Manual > eFuse Controller (eFuse) > System Parameter coding_scheme [PDF].

The algorithm operates on a 256 bit key in all cases, 192 bit keys are extended by repeating some bits (details).

- “secure boot signing key” is a standard ECDSA public/private key pair (see Image Signing Algorithm) in PEM format.
- The public key from this key pair (for signature verification but not signature creation) is compiled into the software bootloader and used to verify the second stage of booting (partition table, app image) before booting continues. The public key can be freely distributed, it does not need to be kept secret.
- The private key from this key pair must be securely kept private, as anyone who has this key can authenticate to any bootloader that is configured with secure boot and the matching public key.

4.26.4 Bootloader Size

Enabling Secure boot and/or flash encryption will increase the size of bootloader, which might require updating partition table offset. See 引导加载程序大小.

4.26.5 How To Enable Secure Boot

1. Open the Project Configuration Menu, navigate to “Secure Boot Configuration” and select the option “One-time Flash”. (To understand the alternative “Reflashable” choice, see Re-Flashable Software Bootloader.)
2. Select a name for the secure boot signing key. This option will appear after secure boot is enabled. The file can be anywhere on your system. A relative path will be evaluated from the project directory. The file does not need to exist yet.
3. Set other menuconfig options (as desired). Pay particular attention to the “Bootloader Config” options, as you can only flash the bootloader once. Then exit menuconfig and save your configuration.

4. The first time you run `make`, if the signing key is not found then an error message will be printed with a command to generate a signing key via `espsecure.py generate_signing_key`.

**Important:** A signing key generated this way will use the best random number source available to the OS and its Python installation (`/dev/urandom` on OSX/Linux and `CryptGenRandom()` on Windows). If this random number source is weak, then the private key will be weak.

**Important:** For production environments, we recommend generating the keypair using openssl or another industry standard encryption program. See *Generating Secure Boot Signing Key* for more details.

5. Run `idf.py bootloader` to build a secure boot enabled bootloader. The build output will include a prompt for a flashing command, using `esptool.py write_flash`.

6. When you’re ready to flash the bootloader, run the specified command (you have to enter it yourself, this step is not performed by make) and then wait for flashing to complete. **Remember this is a one time flash, you can’t change the bootloader after this!**

7. Run `idf.py flash` to build and flash the partition table and the just-built app image. The app image will be signed using the signing key you generated in step 4.

备注：`idf.py flash` doesn’t flash the bootloader if secure boot is enabled.

8. Reset the ESP32 and it will boot the software bootloader you flashed. The software bootloader will enable secure boot on the chip, and then it verifies the app image signature and boots the app. You should watch the serial console output from the ESP32 to verify that secure boot is enabled and no errors have occurred due to the build configuration.

备注：Secure boot won’t be enabled until after a valid partition table and app image have been flashed. This is to prevent accidents before the system is fully configured.

备注：If the ESP32 is reset or powered down during the first boot, it will start the process again on the next boot.

9. On subsequent boots, the secure boot hardware will verify the software bootloader has not changed (using the secure bootloader key) and then the software bootloader will verify the signed partition table and app image (using the public key portion of the secure boot signing key).

### 4.26.6 Re-Flashable Software Bootloader

Configuration “Secure Boot: One-Time Flash” is the recommended configuration for production devices. In this mode, each device gets a unique key that is never stored outside the device.

However, an alternative mode *Secure Boot: Reflashable* is also available. This mode allows you to supply a binary key file that is used for the secure bootloader key. As you have the key file, you can generate new bootloader images and secure boot digests for them.

In the esp-idf build process, this 256-bit key file is derived from the ECDSA app signing key generated by the user (see the *Generating Secure Boot Signing Key* step below). This private key’s SHA-256 digest is used as the secure bootloader key in eFuse (as-is for Coding Scheme None, or truncate to 192 bytes for 3/4 Encoding). This is a convenience so you only need to generate/protect a single private key.
Chapter 4. API

Although it’s possible, we strongly recommend not generating one secure boot key and flashing it to every device in a production environment. The “One-Time Flash” option is recommended for production environments.

To enable a reflashable bootloader:

1. In the Project Configuration Menu, select “Bootloader Config” -> CONFIG_SECURE_BOOT -> CONFIG_SECURE_BOOT_V1_ENABLED -> CONFIG_SECURE_BOOTLOADER_MODE -> Reflashable.
2. If necessary, set the CONFIG_SECURE_BOOTLOADER_KEY_ENCODING based on the coding scheme used by the device. The coding scheme is shown in the Features line when esptool.py connects to the chip, or in the espefuse.py summary output.
3. Please follow the steps shown in Generating Secure Boot Signing Key to generate signing key. Path of the generated key file must be specified in “Secure Boot Configuration” menu.
4. Run idf.py bootloader. A binary key file will be created, derived from the private key that is used for signing. Two sets of flashing steps will be printed - the first set of steps includes an espefuse.py burn_key secure_boot_v1 path_to/secure-bootloader-key-xxx.bin command which is used to write the bootloader key to efuse. (Flashing this key is a one-time-only process.) The second set of steps can be used to reflash the bootloader with a pre-calculated digest (generated during the build process).
5. Resume from Step 6 of the one-time flashing process, to flash the bootloader and enable secure boot. Watch the console log output closely to ensure there were no errors in the secure boot configuration.

4.26.7 Generating Secure Boot Signing Key

The build system will prompt you with a command to generate a new signing key via espsecure.py generate_signing_key. This uses the python-ecdsa library, which in turn uses Python’s os.urandom() as a random number source.

The strength of the signing key is proportional to (a) the random number source of the system, and (b) the correctness of the algorithm used. For production devices, we recommend generating signing keys from a system with a quality entropy source, and using the best available EC key generation utilities.

For example, to generate a signing key using the openssl command line:

```
`openssl ecparam -name prime256v1 -genkey -noout -out my_secure_bootSigning_key.pem`
```

Remember that the strength of the secure boot system depends on keeping the signing key private.

4.26.8 Remote Signing of Images

For production builds, it can be good practice to use a remote signing server rather than have the signing key on the build machine (which is the default esp-idf secure boot configuration). The espsecure.py command line program can be used to sign app images & partition table data for secure boot, on a remote system.

To use remote signing, disable the option “Sign binaries during build”. The private signing key does not need to be present on the build system. However, the public (signature verification) key is required because it is compiled into the bootloader (and can be used to verify image signatures during OTA updates.

To extract the public key from the private key:

```
espsecure.py extract_public_key --keyfile PRIVATE_SIGNING_KEY PUBLIC_VERIFICATION_KEY
```

The path to the public signature verification key needs to be specified in the menuconfig under “Secure boot public signature verification key” in order to build the secure bootloader.

After the app image and partition table are built, the build system will print signing steps using espsecure.py:
4.26.9 Secure Boot Best Practices

- Generate the signing key on a system with a quality source of entropy.
- Keep the signing key private at all times. A leak of this key will compromise the secure boot system.
- Do not allow any third party to observe any aspects of the key generation or signing process using espsecure.py. Both processes are vulnerable to timing or other side-channel attacks.
- Enable all secure boot options in the Secure Boot Configuration. These include flash encryption, disabling of JTAG, disabling BASIC ROM interpreter, and disabling the UART bootloader encrypted flash access.
- Use secure boot in combination with flash encryption to prevent local readout of the flash contents.

4.26.10 Technical Details

The following sections contain low-level reference descriptions of various secure boot elements:

Secure Boot Hardware Support

The first stage of secure boot verification (checking the software bootloader) is done via hardware. The ESP32’s Secure Boot support hardware can perform three basic operations:

1. Generate a random sequence of bytes from a hardware random number generator.
2. Generate a digest from data (usually the bootloader image from flash) using a key stored in Efuse block 2. The key in Efuse can (& should) be read/write protected, which prevents software access. For full details of this algorithm see Secure Bootloader Digest Algorithm. The digest can only be read back by software if Efuse ABS_DONE_0 is not burned (ie still 0).
3. Generate a digest from data (usually the bootloader image from flash) using the same algorithm as step 2 and compare it to a pre-calculated digest supplied in a buffer (usually read from flash offset 0x0). The hardware returns a true/false comparison without making the digest available to software. This function is available even when Efuse ABS_DONE_0 is burned.

Secure Bootloader Digest Algorithm

Starting with an “image” of binary data as input, this algorithm generates a digest as output. The digest is sometimes referred to as an “abstract” in hardware documentation.

For a Python version of this algorithm, see the espsecure.py tool in the components/esptool_py directory (specifically, the digest_secure_bootloader command).

Items marked with (^) are to fulfill hardware restrictions, as opposed to cryptographic restrictions.

1. Read the AES key from efuse block 2, in reversed byte order. If Coding Scheme is set to 3/4 Encoding, extend the 192 bit key to 256 bits using the same algorithm described in Flash加密算法. (^)
2. Prefix the image with a 128 byte randomly generated IV.
3. If the image length is not modulo 128, pad the image to a 128 byte boundary with 0xFF. (^) 4. For each 16 byte plaintext block of the input image: - Reverse the byte order of the plaintext input block (^) - Apply AES256 in ECB mode to the plaintext block. - Reverse the byte order of the ciphertext output block. (^) - Append to the overall ciphertext output.
5. Byte-swap each 4 byte word of the ciphertext (^)
6. Calculate SHA-512 of the ciphertext.
Output digest is 192 bytes of data: The 128 byte IV, followed by the 64 byte SHA-512 digest.

**Image Signing Algorithm**


- Curve is NIST256p (openssl calls this curve “prime256v1”, it is also sometimes called secp256r1).
- Hash function is SHA256.
- Key format used for storage is PEM.
  - In the bootloader, the public key (for signature verification) is flashed as 64 raw bytes.
- Image signature is 68 bytes - a 4 byte version word (currently zero), followed by a 64 bytes of signature data. These 68 bytes are appended to an app image or partition table data.

**Manual Commands**

Secure boot is integrated into the esp-idf build system, so make will automatically sign an app image if secure boot is enabled. `idf.py bootloader` will produce a bootloader digest if menuconfig is configured for it.

However, it is possible to use the `espsecure.py` tool to make standalone signatures and digests.

To sign a binary image:

```
espsecure.py sign_data --keyfile ./my_signing_key.pem --output ./image_signed.bin
```

Keyfile is the PEM file containing an ECDSA private signing key.

To generate a bootloader digest:

```
espsecure.py digest_secure_bootloader --keyfile ./securebootkey.bin --output ./bootloader-digest.bin build/bootloader/bootloader.bin
```

Keyfile is the 32 byte raw secure boot key for the device.

The output of the `espsecure.py digest_secure_bootloader` command is a single file which contains both the digest and the bootloader appended to it. To flash the combined digest plus bootloader to the device:

```
esptool.py write_flash 0x0 bootloader-digest.bin
```

### 4.26.11 Secure Boot & Flash Encryption

If secure boot is used without Flash Encryption, it is possible to launch “time-of-check to time-of-use” attack, where flash contents are swapped after the image is verified and running. Therefore, it is recommended to use both the features together.

### 4.26.12 Signed App Verification Without Hardware Secure Boot

The integrity of apps can be checked even without enabling the hardware secure boot option. This option uses the same app signature scheme as hardware secure boot, but unlike hardware secure boot it does not prevent the bootloader from being physically updated. This means that the device can be secured against remote network access, but not physical access. Compared to using hardware Secure Boot this option is much simpler to implement. See [How To Enable Signed App Verification](#) for step by step instructions.

An app can be verified on update and, optionally, be verified on boot.

- Verification on update: When enabled, the signature is automatically checked whenever the esp_ota_ops.h APIs are used for OTA updates. If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option still adds significant security against network-based attackers by preventing spoofing of OTA updates.
Verification on boot: When enabled, the bootloader will be compiled with code to verify that an app is signed before booting it. If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option doesn’t add significant security by itself so most users will want to leave it disabled.

How To Enable Signed App Verification

1. Open Project Configuration Menu -> Security features -> Enable CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT
2. “Bootloader verifies app signatures” can be enabled, which verifies app on boot.
3. By default, “Sign binaries during build” will be enabled on selecting “Require signed app images” option, which will sign binary files as a part of build process. The file named in “Secure boot private signing key” will be used to sign the image.
4. If you disable “Sign binaries during build” option then you’ll have to enter path of a public key file used to verify signed images in “Secure boot public signature verification key”. In this case, private signing key should be generated by following instructions in Generating Secure Boot Signing Key; public verification key and signed images should be generated by following instructions in Remote Signing of Images.

4.26.13 Advanced Features

JTAG Debugging

By default, when Secure Boot is enabled then JTAG debugging is disabled via eFuse. The bootloader does this on first boot, at the same time it enables Secure Boot.

See JTAG 与闪存安全和安全引导 for more information about using JTAG Debugging with either Secure Boot or signed app verification enabled.

4.27 Secure Boot V2

Important: This document is about Secure Boot V2, supported on the following chips: ESP32 (ECO3 onwards), ESP32-S2, ESP32-S3, ESP32-C3 (ECO3 onwards), and ESP32-C2. Except for ESP32, it is the only supported Secure Boot scheme.

For ESP32 before ECO3, refer to Secure Boot. It is recommended that users use Secure Boot V2 if they have a chip version that supports it. Secure Boot V2 is safer and more flexible than Secure Boot V1.

Secure Boot V2 uses RSA-PSS based app and bootloader verification. This document can also be used as a reference for signing apps using the RSA-PSS scheme without signing the bootloader.

Secure Boot V2 and RSA scheme (App Signing Scheme) options are available for ESP32 from ECO3 onwards. To use these options in menuconfig, set CONFIG_ESP32_REV_MIN greater than or equal to Rev 3.

4.27.1 Background

Secure Boot protects a device from running any unauthorized (i.e., unsigned) code by checking that each piece of software that is being booted is signed. On an ESP32, these pieces of software include the second stage bootloader and each application binary. Note that the first stage bootloader does not require signing as it is ROM code thus cannot be changed.

The Secure Boot process on the ESP32 involves the following steps:

1. When the first stage bootloader loads the second stage bootloader, the second stage bootloader’s RSA-PSS signature is verified. If the verification is successful, the second stage bootloader is executed.
2. When the second stage bootloader loads a particular application image, the application’s RSA-PSS signature is verified. If the verification is successful, the application image is executed.

### 4.27.2 Advantages

- The RSA-PSS public key is stored on the device. The corresponding RSA-PSS private key is kept at a secret place and is never accessed by the device.
- Only one public key can be generated and stored in the chip during manufacturing.
- Same image format and signature verification method is applied for applications and software bootloader.
- No secrets are stored on the device. Therefore, it is immune to passive side-channel attacks (timing or power analysis, etc.)

### 4.27.3 Secure Boot V2 Process

This is an overview of the Secure Boot V2 Process. Instructions how to enable Secure Boot are supplied in section *How To Enable Secure Boot V2*.

Secure Boot V2 verifies the bootloader image and application binary images using a dedicated *signature block*.

Each image has a separately generated signature block which is appended to the end of the image.

Only one signature block can be appended to the bootloader or application image in ESP32 ECO3.

Each signature block contains a signature of the preceding image as well as the corresponding RSA-3072 public key.

For more details about the format, refer to *Signature Block Format*. A digest of the RSA-3072 public key is stored in the eFuse.

The application image is not only verified on every boot but also on each over the air (OTA) update. If the currently selected OTA app image cannot be verified, the bootloader will fall back and look for another correctly signed application image.

The Secure Boot V2 process follows these steps:

1. On startup, the ROM code checks the Secure Boot V2 bit in the eFuse. If Secure Boot is disabled, a normal boot will be executed. If Secure Boot is enabled, the boot will proceed according to the following steps.
2. The ROM code verifies the bootloader’s signature block (*Verifying a Signature Block*). If this fails, the boot process will be aborted.
3. The ROM code verifies the bootloader image using the raw image data, its corresponding signature block(s), and the eFuse (*Verifying an Image*). If this fails, the boot process will be aborted.
4. The ROM code executes the bootloader.
5. The bootloader verifies the application image’s signature block (*Verifying a Signature Block*). If this fails, the boot process will be aborted.
6. The bootloader verifies the application image using the raw image data, its corresponding signature blocks and the eFuse (*Verifying an Image*). If this fails, the boot process will be aborted. If the verification fails but another application image is found, the bootloader will then try to verify that other image using steps 5 to 7. This repeats until a valid image is found or no other images are found.
7. The bootloader executes the verified application image.

### 4.27.4 Signature Block Format

The bootloader and application images are padded to the next 4096 byte boundary, thus the signature has a flash sector of its own. The signature is calculated over all bytes in the image including the padding bytes.

The content of each signature block is shown in the following table:
### Table 28: Content of a Signature Block

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Magic byte</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Version number byte (currently 0x02), 0x01 is for Secure Boot V1.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Padding bytes, Reserved. Should be zero.</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>SHA-256 hash of only the image content, not including the signature block.</td>
</tr>
<tr>
<td>36</td>
<td>384</td>
<td>RSA Public Modulus used for signature verification. (value ‘n’ in RFC8017).</td>
</tr>
<tr>
<td>420</td>
<td>4</td>
<td>RSA Public Exponent used for signature verification (value ‘e’ in RFC8017).</td>
</tr>
<tr>
<td>424</td>
<td>384</td>
<td>Pre-calculated R, derived from ‘n’.</td>
</tr>
<tr>
<td>808</td>
<td>4</td>
<td>Pre-calculated M, derived from ‘n’.</td>
</tr>
<tr>
<td>812</td>
<td>384</td>
<td>RSA-PSS Signature result (section 8.1.1 of RFC8017) of image content, computed using following PSS parameters: SHA256 hash, MGF1 function, salt length 32 bytes, default trailer field (0xBC).</td>
</tr>
<tr>
<td>1196</td>
<td>4</td>
<td>CRC32 of the preceding 1196 bytes.</td>
</tr>
<tr>
<td>1200</td>
<td>16</td>
<td>Zero padding to length 1216 bytes.</td>
</tr>
</tbody>
</table>

备注：R和M’是用于硬件辅助蒙哥蒙特Montgomery乘法。

The remainder of the signature sector is erased flash (0xFF) which allows writing other signature blocks after previous signature block.

#### 4.27.5 Verifying a Signature Block

A signature block is “valid” if the first byte is 0xe7 and a valid CRC32 is stored at offset 1196. Otherwise it’s invalid.

#### 4.27.6 Verifying an Image

An image is “verified” if the public key stored in any signature block is valid for this device, and if the stored signature is valid for the image data read from flash.

1. Compare the SHA-256 hash digest of the public key embedded in the bootloader’s signature block with the digest(s) saved in the eFuses. If public key’s hash doesn’t match any of the hashes from the eFuses, the verification fails.
2. Generate the application image digest and match it with the image digest in the signature block. If the digests don’t match, the verification fails.
3. Use the public key to verify the signature of the bootloader image, using RSA-PSS (section 8.1.2 of RFC8017) with the image digest calculated in step (2) for comparison.

#### 4.27.7 Bootloader Size

Enabling Secure boot and/or flash encryption will increase the size of bootloader, which might require updating partition table offset. See 阅读加载程序大小.

#### 4.27.8 eFuse usage

ESP32-ECO3:

- ABS_DONE_1 - Enables Secure Boot protection on boot.
Chapter 4. API

- BLK2 - Stores the SHA-256 digest of the public key. SHA-256 hash of public key modulus, exponent, pre-calculated R & M’ values (represented as 776 bytes - offsets 36 to 812 - as per the Signature Block Format) is written to an eFuse key block. The write-protection bit must be set, but the read-protection bit must not.

The key(s) must be readable in order to give software access to it. If the key(s) is read-protected then the software reads the key(s) as all zeros and the signature verification process will fail, and the boot process will be aborted.

4.27.9 How To Enable Secure Boot V2

1. Open the Project Configuration Menu, in “Security features” set “Enable hardware Secure Boot in bootloader” to enable Secure Boot.

2. For ESP32, Secure Boot V2 is available only ESP32 ECO3 onwards. To view the “Secure Boot V2” option the chip revision should be changed to revision 3 (ESP32- ECO3). To change the chip revision, set “Minimum Supported ESP32 Revision” to Rev 3 in “Component Config” -> “ESP32- Specific”.

3. Specify the path to Secure Boot signing key, relative to the project directory.

4. Select the desired UART ROM download mode in “UART ROM download mode”. By default the UART ROM download mode has been kept enabled in order to prevent permanently disabling it in the development phase, this option is a potentially insecure option. It is recommended to disable the UART download mode for better security.

5. Set other menuconfig options (as desired). Then exit menuconfig and save your configuration.

6. The first time you run idf.py build, if the signing key is not found then an error message will be printed with a command to generate a signing key via espsecure.py generate_signing_key.

重要: A signing key generated this way will use the best random number source available to the OS and its Python installation (/dev/urandom on OSX/Linux and CryptGenRandom() on Windows). If this random number source is weak, then the private key will be weak.

重要: For production environments, we recommend generating the key pair using openssl or another industry standard encryption program. See Generating Secure Boot Signing Key for more details.

7. Run idf.py bootloader to build a Secure Boot enabled bootloader. The build output will include a prompt for a flashing command, using esptool.py write_flash.

8. When you’re ready to flash the bootloader, run the specified command (you have to enter it yourself, this step is not performed by the build system) and then wait for flashing to complete.

9. Run idf.py flash to build and flash the partition table and the just-built app image. The app image will be signed using the signing key you generated in step 6.

备注: idf.py flash doesn’t flash the bootloader if Secure Boot is enabled.

10. Reset the ESP32 and it will boot the software bootloader you flashed. The software bootloader will enable Secure Boot on the chip, and then it verifies the app image signature and boots the app. You should watch the serial console output from the ESP32 to verify that Secure Boot is enabled and no errors have occurred due to the build configuration.

备注: Secure boot won’t be enabled until after a valid partition table and app image have been flashed. This is to prevent accidents before the system is fully configured.

备注: If the ESP32 is reset or powered down during the first boot, it will start the process again on the next boot.
11. On subsequent boots, the Secure Boot hardware will verify the software bootloader has not changed and the software bootloader will verify the signed app image (using the validated public key portion of its appended signature block).

### 4.27.10 Restrictions after Secure Boot is enabled

- Any updated bootloader or app will need to be signed with a key matching the digest already stored in eFuse.
- After Secure Boot is enabled, no further eFuses can be read protected. (If Flash encryption is enabled then the bootloader will ensure that any flash encryption key generated on first boot will already be read protected.) If `CONFIG_SECURE_BOOT_INSECURE` is enabled then this behavior can be disabled, but this is not recommended.

### 4.27.11 Generating Secure Boot Signing Key

The build system will prompt you with a command to generate a new signing key via `espsecure.py generate_signing_key`.

The `--version 2` parameter will generate the RSA 3072 private key for Secure Boot V2.

The strength of the signing key is proportional to (a) the random number source of the system, and (b) the correctness of the algorithm used. For production devices, we recommend generating signing keys from a system with a quality entropy source, and using the best available RSA-PSS key generation utilities.

For example, to generate a signing key using the openssl command line:

```
openssl genrsa -out my_secure_boot_signing_key.pem 3072
```

Remember that the strength of the Secure Boot system depends on keeping the signing key private.

### 4.27.12 Remote Signing of Images

For production builds, it can be good practice to use a remote signing server rather than have the signing key on the build machine (which is the default esp-idf Secure Boot configuration). The `espsecure.py` command line program can be used to sign app images & partition table data for Secure Boot, on a remote system.

To use remote signing, disable the option “Sign binaries during build”. The private signing key does not need to be present on the build system.

After the app image and partition table are built, the build system will print signing steps using `espsecure.py`:

```
espsecure.py sign_data --version 2 --keyfile PRIVATE_SIGNING_KEY BINARY_FILE
```

The above command appends the image signature to the existing binary. You can use the `--output` argument to write the signed binary to a separate file:

```
espsecure.py sign_data --version 2 --keyfile PRIVATE_SIGNING_KEY --output SIGNED_--BINARY_FILE BINARY_FILE
```

### 4.27.13 Secure Boot Best Practices

- Generate the signing key on a system with a quality source of entropy.
- Keep the signing key private at all times. A leak of this key will compromise the Secure Boot system.
- Do not allow any third party to observe any aspects of the key generation or signing process using `espsecure.py`. Both processes are vulnerable to timing or other side-channel attacks.
- Enable all Secure Boot options in the Secure Boot Configuration. These include flash encryption, disabling of JTAG, disabling BASIC ROM interpreter, and disabling the UART bootloader encrypted flash access.
- Use Secure Boot in combination with flash encryption to prevent local readout of the flash contents.
4.27.14 Technical Details

The following sections contain low-level reference descriptions of various Secure Boot elements:

Manual Commands

Secure boot is integrated into the esp-idf build system, so `idf.py build` will sign an app image and `idf.py bootloader` will produce a signed bootloader if secure signed binaries on build is enabled.

However, it is possible to use the `espsecure.py` tool to make standalone signatures and digests.

To sign a binary image:

```
espsecure.py sign_data --version 2 --keyfile ./my_signing_key.pem --output ./image_\n\n\nimage_signed.bin image_unsigned.bin
```

Keyfile is the PEM file containing an RSA-3072 private signing key.

4.27.15 Secure Boot & Flash Encryption

If Secure Boot is used without Flash Encryption, it is possible to launch “time-of-check to time-of-use” attack, where flash contents are swapped after the image is verified and running. Therefore, it is recommended to use both the features together.

4.27.16 Signed App Verification Without Hardware Secure Boot

The Secure Boot V2 signature of apps can be checked on OTA update, without enabling the hardware Secure Boot option. This option uses the same app signature scheme as Secure Boot V2, but unlike hardware Secure Boot it does not prevent an attacker who can write to flash from bypassing the signature protection.

This may be desirable in cases where the delay of Secure Boot verification on startup is unacceptable, and/or where the threat model does not include physical access or attackers writing to bootloader or app partitions in flash.

In this mode, the public key which is present in the signature block of the currently running app will be used to verify the signature of a newly updated app. (The signature on the running app isn’t verified during the update process, it’s assumed to be valid.) In this way the system creates a chain of trust from the running app to the newly updated app.

For this reason, it’s essential that the initial app flashed to the device is also signed. A check is run on app startup and the app will abort if no signatures are found. This is to try and prevent a situation where no update is possible. The app should have only one valid signature block in the first position. Note again that, unlike hardware Secure Boot V2, the signature of the running app isn’t verified on boot. The system only verifies a signature block in the first position and ignores any other appended signatures.

备注： In general, it’s recommended to use full hardware Secure Boot unless certain that this option is sufficient for application security needs.

How To Enable Signed App Verification

1. Open Project Configuration Menu -> Security features
2. Ensure App Signing Scheme is RSA. For ESP32 ECO3 chip, select `CONFIG_ESP32_REV_MIN` to Rev 3 to get RSA option available
3. Enable `CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT`
4. By default, “Sign binaries during build” will be enabled on selecting “Require signed app images” option, which will sign binary files as a part of build process. The file named in “Secure boot private signing key” will be used to sign the image.
5. If you disable “Sign binaries during build” option then all app binaries must be manually signed by following instructions in *Remote Signing of Images*.

警告: It is very important that all apps flashed have been signed, either during the build or after the build.

### 4.27.17 Advanced Features

**JTAG Debugging**

By default, when Secure Boot is enabled then JTAG debugging is disabled via eFuse. The bootloader does this on first boot, at the same time it enables Secure Boot.

See *JTAG与闪存加密和安全引导* for more information about using JTAG Debugging with either Secure Boot or signed app verification enabled.

### 4.28 片外 RAM

#### 4.28.1 简介

ESP32 提供了好几 KB 的片上 RAM，可以满足大部分需求。但有些场景可能需要更多 RAM，因此 ESP32 另外提供了高达 4 MB 的片外 SPI RAM 存储器供用户使用。片外 RAM 已经集成到内存映射中，在某些范围内与片上 RAM 使用方式相同。

#### 4.28.2 硬件

ESP32 支持与 SPI flash 芯片并联的 SPI PSRAM（伪静态随机存储器）。虽然 ESP32 支持多种类型的 RAM 芯片，但 ESP-IDF 当前仅支持乐鑫品牌的 PSRAM 芯片，如 ESP-PSRAM32、ESP-PSRAM64 等。

**备注**: PSRAM 芯片的工作电压分为 1.8 V 和 3.3 V。其工作电压必须与 flash 的工作电压匹配。请查询您 PSRAM 芯片以及 ESP32 的技术规格书获取准确的工作电压。对于 1.8 V 的 PSRAM 芯片，请确保在启动时将 MTDI 管脚设置为高电平，或者将 ESP32 中的 eFuses 设置为始终使用 1.8 V 的 VDD_SIO 电平，否则有可能会损坏 PSRAM 和/或 flash 芯片。

**备注**: 乐鑫同时提供模组和系统级封装芯片，集成了兼容的 PSRAM 和 flash，可直接用于终端产品 PCB 中。如需了解更多信息，请前往乐鑫官网。

有关将 SoC 或模组管脚连接到片外 PSRAM 芯片的具体细节，请查阅 SoC 或模组技术规格书。

#### 4.28.3 配置片外 RAM

ESP-IDF 完全支持将片外 RAM 集成到您的应用程序中。在启动并完成片外 RAM 初始化后，可以将 ESP-IDF 配置为用多种方式处理片外 RAM:

- 集成片外 RAM 到 ESP32 内存映射
- 添加片外 RAM 到堆内存分配器
- 调用 malloc() 分配片外 RAM (default)
集成片外 RAM 到 ESP32 内存映射

在 CONFIG_SPIRAM_USE 中选择 “Integrate RAM into memory map (集成片外 RAM 到 ESP32 内存映射)” 选项。

这是集成片外 RAM 最基础的设置选项，大多数用户需要用到其他更高级的选项。

ESP-IDF 启动过程中，片外 RAM 被映射到 0x3F800000 起始的数据地址空间（字节可寻址），空间大小正好为 SPI RAM 的大小 (4 MB)。

应用程序可以通过创建指向该区域的指针手动将数据放入片外存储器，同时应用程序全程负责管理片外 SPI RAM，包括调用 Buffe 的使用，防止发生损坏等。

添加片外 RAM 到内存分配器

在 CONFIG_SPIRAM_USE 中选择 “Make RAM allocatable using heap_caps_malloc(...)” 选项。

启用上述选项后，片外 RAM 被映射到地址 0x3F800000，并将这个区域添加到携带 MALLOC_CAP_SPIRAM 标志的内存分配器。

程序如果想从片外存储器分配存储空间，则需要调用 heap_caps_malloc(size, MALLOC_CAP_SPIRAM)，之后可以调用 free() 函数释放这部分存储空间。

调用 malloc() 分配片外 RAM

在 CONFIG_SPIRAM_USE 中选择 “Make RAM allocatable using malloc() as well” 选项，该选项为默认选项。

启用此选项后，片外存储器将被添加到内存分配程序（与上一选项相同），同时也将被添加到由标准 malloc() 函数返回的 RAM 中。

应用程序因此可以使用片外 RAM，无需重写代码就能使用 heap_caps_malloc(..., MALLOC_CAP_SPIRAM)。

如果某次内存分配偏向于片外存储器，您也可以使用 CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL 设置分配空间的大小阈值，控制分配结果：

- 如果分配的空间小于阈值，分配程序将首先选择内部存储器。
- 如果分配的空间等于或大于阈值，分配程序将首先选择外部存储器。

如果优先考虑的内部或外部存储器中没有可用的存储块，分配程序则会选择其他类型存储。

由于有些内存缓冲器仅可在内部存储器中分配，因此需要使用第二个配置项 CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL 定义一个内存内核池，仅限显式的内部存储器分配使用（例如用于 DMA 的存储器）。常规 malloc() 将不会从该池中分配，但可以使用 MALLOC_CAP_DMA 和 MALLOC_CAP_INTERNAL 标志从该池中分配存储器。

允许.bss 段放入片外存储器

通过勾选 CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY 启用该选项。

启用该选项后，从 0x3F800000 起始的地址空间将用于存储来自 lwip, net80211, libpp 和 bluedroid ESP-IDF 库中初始化的数据（BSS 段）。

通过将宏 EXT_RAM_BSS_ATTR 应用于任何静态声明（初始化为非零值），可以将附加数据从内部 BSS 段移到片外 RAM。

也可以使用链接器片段方案 extram_bss 将组件即库的 BSS 段放到片外 RAM 中。

启用此选项可以减少 BSS 段占用的内部静态存储。

Espressif Systems 2363 Release v5.1-dev-2066-g7869f4e151
剩余的片外 RAM 也可以通过上述方法添加到堆分配器中。

允许`noinit`段放入片外存储器

通过勾选`CONFIG_SPIRAM_ALLOW_NOINIT_SEG_EXTERNAL_MEMORY`启用该选项。启用该选项后，外部 RAM 中提供的地址空间区域将用于存储未初始化的数据。即使在启动或重新启动期间，放置在该段中的值也不会被初始化或修改。

通过应用`EXT_RAM_NOINIT_ATTR`宏，可以将数据从内部 NOINIT 段移到片外 RAM。剩余的片外 RAM 也可以通过上述方法添加到堆分配器中，具体请参考添加片外 RAM 到堆内存分配器。

### 4.28.4 片外 RAM 使用限制

使用片外 RAM 有下面一些限制：

- flash cache 禁用时（比如，正在写入 flash），片外 RAM 将无法访问；同样，对片外 RAM 的读写操作也将导致 cache 访问异常。出于这个原因，ESP-IDF 不会在片外 RAM 中分配任务堆栈（详见下文）。
- 片外 RAM 不能用于储存 DMA 事务描述符，也不能用作 DMA 读写操作的缓冲区（Buffer）。因此，当片外 RAM 启用时，与 DMA 搭配使用的 Buffer 必须先使用 `heap_caps_malloc(size, MALLOC_CAP_DMA | MALLOC_CAP_INTERNAL)` 进行分配，之后可以调用标准 `free()` 回调释放 Buffer。
- 片外 RAM 与片外 flash 使用相同的 cache 区域，这意味着频繁在片外 RAM 访问的变量可以像在片上 RAM 中一样快速读取和修改。但访问大块数据时（大于 32 KB），cache 空间可能会不足，访问速度将回落到片外 RAM 访问速度。此外，访问大块数据会挤出 flash cache，可能降低代码执行速度。
- 一般来说，片外 RAM 不会用作任务堆栈存储器。`xTaskCreate()` 及类似函数始终会为堆栈和任务 TCB 分配片上储存器。

可以使用`CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY`选将任务堆栈放入片外存储器。这时，必须使用`xTaskCreateStatic()` 指定从片外存储器分配的任务堆栈缓冲区，否则任务堆栈将会从片上存储器分配。

### 4.28.5 初始化失败

默认情况下，片外 RAM 初始化失败将终止 ESP-IDF 启动。如果想禁用此功能，可启用`CONFIG_SPIRAM_IGNORE_NOTFOUND`配置选项。

如果启用`CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY`，忽略失败的选项将无法使用，这是因为当链接时，链接器已经向片外存储器分配标志符。

- Regarding stacks in PSRAM: For tasks not calling on code in ROM in any way, directly or indirectly, the menuconfig option `CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY` will eliminate the check in `xTaskCreateStatic`, allowing a task’s stack to be in external RAM. Using this is not advised, however.

### 4.28.6 Chip revisions

There are some issues with certain revisions of ESP32 that have repercussions for use with external RAM. The issues are documented in the ESP32 ECO document. In particular, ESP-IDF handles the bugs mentioned in the following ways:

ESP32 rev v0

ESP-IDF has no workaround for the bugs in this revision of silicon, and it cannot be used to map external PSRAM into ESP32’s main memory map.
Chapter 4. API

ESP32 rev v1

The bugs in this revision of silicon cause issues if certain sequences of machine instructions operate on external memory. (ESP32 ECO 3.2). As a workaround, the GCC compiler received the flag `-mfix-esp32-psram-cache-issue` to filter these sequences and only output the code that can safely be executed. Enable this flag by checking `CONFIG_SPIRAM_CACHE_WORKAROUND`.

Aside from linking to a recompiled version of Newlib with the additional flag, ESP-IDF also does the following:

- Avoids using some ROM functions
- Allocates static memory for the WiFi stack

ESP32 rev v3

ESP32 revision 3 (“ECO V3”) fixes the PSRAM cache issue found in rev. 1. When `CONFIG_ESP32_REV_MIN` option is set to rev. 3, compiler workarounds related to PSRAM will be disabled. For more information about ESP32 ECO V3, see ESP32 ECO V3 User Guide.

4.29 Thread Local Storage

4.29.1 Overview

Thread-local storage (TLS) is a mechanism by which variables are allocated such that there is one instance of the variable per extant thread. ESP-IDF provides three ways to make use of such variables:

- **FreeRTOS Native API**: ESP-IDF FreeRTOS native API.
- **Pthread API**: ESP-IDF’s pthread API.
- **C11 Standard**: C11 standard introduces special keyword to declare variables as thread local.

4.29.2 FreeRTOS Native API

The ESP-IDF FreeRTOS provides the following API to manage thread local variables:

- `vTaskSetThreadLocalStoragePointer()`
- `pvTaskGetThreadLocalStoragePointer()`
- `vTaskSetThreadLocalStoragePointerAndDelCallback()`

In this case maximum number of variables that can be allocated is limited by `CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS` configuration value. Variables are kept in the task control block (TCB) and accessed by their index. Note that index 0 is reserved for ESP-IDF internal uses.

Using that API user can allocate thread local variables of an arbitrary size and assign them to any number of tasks. Different tasks can have different sets of TLS variables.

If size of the variable is more then 4 bytes then user is responsible for allocating/deallocating memory for it. Variable’s deallocation is initiated by FreeRTOS when task is deleted, but user must provide function (callback) to do proper cleanup.

4.29.3 Pthread API

The ESP-IDF provides the following pthread API to manage thread local variables:

- `pthread_key_create()`
- `pthread_key_delete()`
- `pthread_getspecific()`
- `pthread_setspecific()`
This API has all benefits of the one above, but eliminates some its limits. The number of variables is limited only by size of available memory on the heap. Due to the dynamic nature this API introduces additional performance overhead compared to the native one.

### 4.29.4 C11 Standard

The ESP-IDF FreeRTOS supports thread local variables according to C11 standard (ones specified with `__thread` keyword). For details on this GCC feature please see [https://gcc.gnu.org/onlinedocs/gcc-5.5.0/gcc/Thread-Local.html#Thread-Local](https://gcc.gnu.org/onlinedocs/gcc-5.5.0/gcc/Thread-Local.html#Thread-Local). Storage for that kind of variables is allocated on the task’s stack. Note that area for all such variables in the program will be allocated on the stack of every task in the system even if that task does not use such variables at all. For example ESP-IDF system tasks (like `ipc.timer` tasks etc.) will also have that extra stack space allocated. So this feature should be used with care. There is a tradeoff: C11 thread local variables are quite handy to use in programming and can be accessed using minimal CPU instructions, but this benefit goes with the cost of additional stack usage for all tasks in the system. Due to static nature of variables allocation all tasks in the system have the same sets of C11 thread local variables.

### 4.30 工具

#### 4.30.1 IDF Frontend - `idf.py`

The `idf.py` command-line tool provides a front-end for easily managing your project builds, deployment and debugging, and more. It manages several tools, for example:

- **CMake**, which configures the project to be built
- **Ninja** which builds the project
- **esptool.py** for flashing the target.

The [getting started guide](https://docs.espressif.com/projects/idf/en/latest/ Getting Started.html) contains a brief introduction to how to set up `idf.py` to configure, build, and flash projects.

**重要**: `idf.py` should be run in an ESP-IDF “project” directory, i.e. one containing a `CMakeLists.txt` file. Older style projects with a `Makefile` will not work with `idf.py`.

**Commands**

**Start a new project: create-project**

```
idf.py create-project <project name>
```

This will create a new ESP-IDF project, additionally folder where the project will be created can be specified by the `--path` option.

**Create a new component: create-component**

```
idf.py create-component <component name>
```

This command creates a new component, which will have a minimum set of files necessary for building.

The `-C` option can be used to specify the directory the component will be created in. For more information about components see the [build system page](https://docs.espressif.com/projects/idf/en/latest/ Build System.html).

**Select the Target Chip: set-target**

ESP-IDF supports multiple targets (chips). A full list of supported targets in your version of ESP-IDF can be seen by running `idf.py --list-targets`.

This sets the current project target:
**Chapter 4. API 指南**

```python
idf.py set-target <target>
```

**重要：** `idf.py set-target` 将清除构建目录并重新生成 `sdkconfig` 文件。旧的 `sdkconfig` 文件将保存为 `sdkconfig.old`。

**备注：** `idf.py set-target` 命令等同于以下操作：

1. 清除构建目录 (`idf.py fullclean`)
2. 移动 `sdkconfig` 文件到 `sdkconfig.old` (`mv sdkconfig sdkconfig.old`)
3. 配置新目标的项目 (`idf.py -DIDF_TARGET=esp32 reconfigure`)

也可以通过将所需的 `IDF_TARGET` 作为环境变量（例如 `export IDF_TARGET=esp32s2`）或作为 CMake 变量（例如 `IDF_TARGET=esp32s2`）传递给 CMake 或 `idf.py` 来指定。在大多数情况下，您可以将环境变量设置为一个方便的方法，如果您主要使用一种类型的芯片。

如果未由任何这些方法设置目标，则构建系统将默认设置为 `esp32` 目标。

**Start the graphical configuration tool: menuconfig**

```python
idf.py menuconfig
```

**Build the project: build**

```python
idf.py build
```

运行此命令将构建当前目录中找到的项目。这可能涉及多个步骤：

- 创建构建目录（`-B` 选项）。
- 运行必需的 CMake 配置整个项目，并为构建工具生成构建文件。
- 运行主构建工具（`Ninja` 或 `GNU Make`）。默认情况下，构建工具将自动检测到它，但可以使用 `-G` 选项显式设置。

构建是增量的，如果自上次构建以来没有更改源文件或配置，则没有任何操作。

另外，命令可以使用 `app`、`bootloader` 和 `partition-table` 参数仅构建这些应用程序、固件或分区表。

**Remove the build output: clean** 通过使用以下命令可以删除构建输出文件：

```python
idf.py clean
```

项目将被完全重建。使用此方法不会从构建目录中删除 CMake 配置输出。

**Delete the entire build contents: fullclean**

```python
idf.py fullclean
```
Running this command will delete the entire "build" directory contents. This includes all CMake configuration output. The next time the project is built, CMake will configure it from scratch. Note that this option recursively deletes all files in the build directory, so use with care. Project configuration is not deleted.

**Flash the project: flash**

Running the following command:

```
idf.py flash
```

will automatically build the project if necessary, and then flash it to the target. You can use `-p` and `-b` options to set serial port name and flasher baud rate, respectively.

**备注:** The environment variables `ESPPORT` and `ESPBAUD` can be used to set default values for the `-p` and `-b` options, respectively. Providing these options on the command line overrides the default.

Similarly to the `build` command, the command can be run with `app`, `bootloader` and `partition-table` arguments to flash only the app, bootloader or partition table as applicable.

**Hints on how to resolve errors**

`idf.py` will try to suggest hints on how to resolve errors. It works with a database of hints stored in `tools/idf_py_actions/hints.yml` and the hints will be printed if a match is found for the given error. The monitor, menuconfig, gdb and openocd targets are not supported at the moment by automatic hints on resolving errors.

The `--no-hints` argument of `idf.py` can be used to turn the hints off in case they are not desired.

**Important notes**

Multiple `idf.py` commands can be combined into one. For example, `idf.py -p COM4 clean flash monitor` will clean the source tree, then build the project and flash it to the target before running the serial monitor.

The order of multiple `idf.py` commands on the same invocation is not important, they will automatically be executed in the correct order for everything to take effect (i.e. building before flashing, erasing before flashing, etc.).

For commands that are not known to `idf.py` an attempt to execute them as a build system target will be made.

The command `idf.py` supports **shell autocompletion** for bash, zsh and fish shells.

In order to make **shell autocompletion** supported, please make sure you have at least Python 3.5 and **click** 7.1 or newer (**see also**).

To enable autocompletion for `idf.py` use the `export` command (**see this**). Autocompletion is initiated by pressing the TAB key. Type `idf.py` - and press the TAB key to autocomplete options.

The autocomplete support for PowerShell is planned in the future.

**Advanced Commands**

**Open the documentation: docs**

Using the following command the documentation for the projects target and version will be opened in the browser:

```
idf.py docs
```

**Show size: size**

```
idf.py size
```

Will print app size information including occupied RAM and FLASH and section sizes.
idf.py `size-components`

Similarly, this will print the same information for each component used in the project.

idf.py `size-files`

Will print size information per source file in the project.

**Options**

- `--format` specifies the output format with available options: `text`, `csv`, `json`, default being `text`.
- `--output-file` optionally specifies the name of the file to print the command output to instead of the standard output.

**Reconfigure the project: reconfigure**

idf.py `reconfigure`

This command re-runs `CMake` even if it doesn’t seem to need re-running. This isn’t necessary during normal usage, but can be useful after adding/removing files from the source tree, or when modifying CMake cache variables. For example, `idf.py -DNAME='VALUE' reconfigure` can be used to set variable `NAME` in CMake cache to value `VALUE`.

**Clean the python byte code: python-clean**

Generated python byte code can be deleted from the IDF directory using:

idf.py `python-clean`

The byte code may cause issues when switching between IDF and Python versions. It is advised to run this target after switching versions of Python.

**Global Options**

To list all available root level options, run `idf.py --help`. To list options that are specific for a subcommand, run `idf.py <command> --help`, for example `idf.py monitor --help`. Here is a list of some useful options:

- `--C <dir>` allows overriding the project directory from the default current working directory.
- `--B <dir>` allows overriding the build directory from the default build subdirectory of the project directory.
- `--ccache` flag can be used to enable CCache when compiling source files, if the CCache tool is installed. This can dramatically reduce some build times.

Note that some older versions of CCache may exhibit bugs on some platforms, so if files are not rebuilt as expected then try disabling CCache and build again. CCache can be enabled by default by setting the `IDF_CCACHE_ENABLE` environment variable to a non-zero value.

- `--v` flag causes both `idf.py` and the build system to produce verbose build output. This can be useful for debugging build problems.
- `--cmake-warn-uninitialized` (or `-w`) will cause CMake to print uninitialized variable warnings found in the project directory only. This only controls CMake variable warnings inside CMake itself, not other types of build warnings. This option can also be set permanently by setting the `IDF_CMAKE_WARN_UNINITIALIZED` environment variable to a non-zero value.
- `--no-hints` flag to disable hints on resolving errors and disable capturing output.
4.30.2 IDF Docker Image

IDF Docker image (espressif/idf) is intended for building applications and libraries with specific versions of ESP-IDF, when doing automated builds.

The image contains:

- Common utilities such as git, wget, curl, zip.
- Python 3.7 or newer.
- A copy of a specific version of ESP-IDF (see below for information about versions). IDF_PATH environment variable is set, and points to ESP-IDF location in the container.
- All the build tools required for the specific version of ESP-IDF: CMake, ninja, cross-compiler toolchains, etc.
- All Python packages required by ESP-IDF are installed in a virtual environment.

The image entrypoint sets up PATH environment variable to point to the correct version of tools, and activates the Python virtual environment. As a result, the environment is ready to use the ESP-IDF build system.

The image can also be used as a base for custom images, if additional utilities are required.

Tags

Multiple tags of this image are maintained:

- latest: tracks master branch of ESP-IDF
- vX.Y: corresponds to ESP-IDF release vX.Y
- release-vX.Y: tracks release/vX.Y branch of ESP-IDF

This feature was introduced before this feature was introduced do not have corresponding Docker image versions. You can check the up-to-date list of available tags at https://hub.docker.com/r/espressif/idf/tags.

Usage

Setting up Docker Before using the espressif/idf Docker image locally, make sure you have Docker installed. Follow the instructions at https://docs.docker.com/install/, if it is not installed yet.

If using the image in CI environment, consult the documentation of your CI service on how to specify the image used for the build process.

Building a project with CMake In the project directory, run:

```shell
docker run --rm -v $PWD:/project -w /project espressif/idf idf.py build
```

The above command explained:

- docker run: runs a Docker image. It is a shorter form of the command docker container run.
- --rm: removes the container when the build is finished
- -v $PWD:/project: mounts the current directory on the host ($PWD) as /project directory in the container
- espressif/idf: uses Docker image espressif/idf with tag latest (implicitly added by Docker when no tag is specified)
- idf.py build: runs this command inside the container

To build with a specific docker image tag, specify it as espressif/idf:TAG, for example:

```shell
docker run --rm -v $PWD:/project -w /project espressif/idf:release-v4.4 idf.py --build
```

You can check the up-to-date list of available tags at https://hub.docker.com/r/espressif/idf/tags.
Using the image interactively  It is also possible to do builds interactively, to debug build issues or test the automated build scripts. Start the container with `-i -t` flags:

```
docker run --rm -v $PWD:/project -w /project -it espressif/idf
```

Then inside the container, use idf.py as usual:

```
idf.py menuconfig
idf.py build
```

备注: Commands which communicate with the development board, such as idf.py flash and idf.py monitor will not work in the container unless the serial port is passed through into the container. However currently this is not possible with Docker for Windows (https://github.com/docker/for-win/issues/1018) and Docker for Mac (https://github.com/docker/for-mac/issues/900).

Building custom images

The Dockerfile in ESP-IDF repository provides several build arguments which can be used to customize the Docker image:

- **IDF_CLONE_URL**: URL of the repository to clone ESP-IDF from. Can be set to a custom URL when working with a fork of ESP-IDF. Default is https://github.com/espressif/esp-idf.git.
- **IDF_CLONE_BRANCH_OR_TAG**: Name of a git branch or tag use when cloning ESP-IDF. This value is passed to git clone command using the `--branch` argument. Default is master.
- **IDF_CHECKOUT_REF**: If this argument is set to a non-empty value, git checkout $IDF_CHECKOUT_REF command will be performed after cloning. This argument can be set to the SHA of the specific commit to check out, for example if some specific commit on a release branch is desired.
- **IDF_CLONE_SHALLOW**: If this argument is set to a non-empty value, `--depth=1 --shallow-submodules` arguments will be used when performing git clone. This significantly reduces the amount of data downloaded and the size of the resulting Docker image. However, if switching to a different branch in such a “shallow” repository is necessary, an additional git fetch origin <branch> command must be executed first.
- **IDF_INSTALL_TARGETS**: Comma-separated list of IDF targets to install toolchains for, or all to install toolchains for all targets. Selecting specific targets reduces the amount of data downloaded and the size of the resulting Docker image. Default is all.

To use these arguments, pass them via the `--build-arg` command line option. For example, the following command will build a Docker image with a shallow clone of ESP-IDF v4.4.1 and tools for ESP32-C3, only:

```
docker build -t idf-custom:v4.4.1-esp32c3
  --build-arg IDF_CLONE_BRANCH_OR_TAG=v4.4.1
  --build-arg IDF_CLONE_SHA=1
  --build-arg IDF_INSTALL_TARGETS=esp32c3
tools/docker
```

4.30.3 IDF Windows Installer

Command-line parameters

Windows Installer esp-idf-tools-setup provides the following command-line parameters:

- `/CONFIG=[PATH]` - Path to ini configuration file to override default configuration of the installer. Default: config.ini.
- `/GITRECURSIVE=[yes|no]` - Clone recursively all git repository submodules. Default: yes
Chapter 4. API

Unattended installation

The unattended installation of IDF can be achieved by following command-line parameters:

```
esp-idf-tools-setup-x.x.exe /VERYSILENT /SUPPRESSMSGBOXES /SP- /NOCANCEL
```

The installer detaches its process from the command-line. Waiting for installation to finish could be achieved by following PowerShell script:

```
$InstallerProcess = Get-Process esp-idf-tools-setup
Wait-Process -Id $InstallerProcess.id
```

Custom Python and custom location of Python wheels

The IDF installer is using by default embedded Python with reference to Python Wheel mirror. Following parameters allows to select custom Python and custom location of Python wheels:

```
esp-idf-tools-setup-x.x.exe /USEEMBEDDEDPYTHON=no /PYTHONWHEELSURL=https://pypi.org/simple/
```

4.30.4 IDF Component Manager

The IDF Component manager is a tool that downloads dependencies for any ESP-IDF CMake project. The download happens automatically during a run of CMake. It can source components either from the component registry or from a git repository.

A list of components can be found on https://components.espressif.com/

Using with a project

Dependencies for each component in the project are defined in a separate manifest file named idf_component.yml placed in the root of the component. The manifest file template can be created for a component by running
idf.py create-manifest --component=my_component. When a new manifest is added to one of the components in the project it’s necessary to reconfigure it manually by running idf.py reconfigure. Then build will track changes in idf_component.yml manifests and automatically triggers CMake when necessary.

There is an example application: example/build_system/cmake/component_manager that uses components installed by the component manager.

It’s not necessary to have a manifest for components that don’t need any managed dependencies.

When CMake configures the project (e.g. idf.py reconfigure) component manager does a few things:

- Processes idf_component.yml manifests for every component in the project and recursively solves dependencies
- Creates a dependencies.lock file in the root of the project with a full list of dependencies
- Downloads all dependencies to the managed_components directory

The lock-file dependencies.lock and content of managed_components directory is not supposed to be modified by a user. When the component manager runs it always make sure they are up to date. If these files were accidentally modified it’s possible to re-run the component manager by triggering CMake with idf.py reconfigure

**Defining dependencies in the manifest**

```yaml
dependencies:
  # Required IDF version
  idf: ">=4.1"
  # Defining a dependency from the registry:
  # https://components.espressif.com/component/example/cmp
  example/cmp: ">=1.0.0"

  # # Other ways to define dependencies
  # # For components maintained by Espressif only name can be used.
  # # Same as `espressif/cmp`
  # component: "~1.0.0"
  # # Or in a longer form with extra parameters
  # component2:
  # version: ">=2.0.0"
  # # For transient dependencies `public` flag can be set.
  # # `public` flag doesn't affect the `main` component.
  # # All dependencies of `main` are public by default.
  # public: true
  # # For components hosted on non-default registry:
  # service_url: "https://componentregistry.company.com"
  # # For components in git repository:
  # test_component:
  # path: test_component
  # git: ssh://git@gitlab.com/user/components.git
  # # For test projects during component development
  # # components can be used from a local directory
  # # with relative or absolute path
  # some_local_component:
  # path: ../../projects/component
```
Disabling the Component Manager

The component manager can be explicitly disabled by setting `IDF_COMPONENT_MANAGER` environment variable to 0.

4.30.5 IDF Clang Tidy

The IDF Clang Tidy is a tool that uses `clang-tidy` to run static analysis on your current app.

| Warning: | This functionality and the toolchain it relies on are still under development. There may be breaking changes before a final release. |

Prerequisites

If you have never run this tool before, take the following steps to get this tool prepared.

1. Run the export scripts (`export.sh`/`export.bat`/...) to set up the environment variables.
2. Run `pip install --upgrade pyclang` to install this plugin. The extra commands would be activated in `idf.py` automatically.
3. Run `idf_tools.py install esp-clang` to install the clang-tidy required binaries

| Note: | This toolchain is still under development. After the final release, you don’t have to install them manually. |

4. Get file from the llvm repository and add the folder of this script to the $PATH. Or you could pass an optional argument `--run-clang-tidy-py` later when you call `idf.py clang-check`. Please don’t forget to make the script executable.

| Note: | This file would be bundled in future toolchain releases. This is a temporary workaround. |

5. Run the export scripts (`export.sh`/`export.bat`/...) again to refresh the environment variables.

Extra Commands

- **clang-check** Run `idf.py clang-check` to re-generate the compilation database and run clang-tidy under your current project folder. The output would be written to `<project_dir>/warnings.txt`.

Run `idf.py clang-check --help` to see the full documentation.

- **clang-html-report**

  1. Run `pip install codereport` to install the additional dependency.

  2. Run `idf.py clang-html-report` to generate an HTML report in folder `<project_dir>/html_report` according to the `warnings.txt`. Please open the `<project_dir>/html_report/index.html` in your browser to check the report.

Bug Report

This tool is hosted in espressif/clang-tidy-runner. If you faced any bugs or have any feature request, please report them via github issues.
4.30.6 Downloadable Tools

ESP-IDF build process relies on a number of tools: cross-compiler toolchains, CMake build system, and others. Installing the tools using an OS-specific package manager (like apt, yum, brew, etc.) is the preferred method when the required version of the tool is available. This recommendation is reflected in the Getting Started guide. For example, on Linux and macOS it is recommended to install CMake using an OS package manager.

However, some of the tools are IDF-specific and are not available in OS package repositories. Furthermore, different versions of ESP-IDF require different versions of the tools to operate correctly. To solve these two problems, ESP-IDF provides a set of scripts for downloading and installing the correct versions of tools, and exposing them in the environment.

The rest of the document refers to these downloadable tools simply as “tools”. Other kinds of tools used in ESP-IDF are:

- Python scripts bundled with ESP-IDF (such as idf.py)
- Python packages installed from PyPI.

The following sections explain the installation method, and provide the list of tools installed on each platform.

Tools metadata file

The list of tools and tool versions required for each platform is located in tools/tools.json. The schema of this file is defined by tools/tools_schema.json. This file is used by tools/idf_tools.py script when installing the tools or setting up the environment variables.

Tools installation directory

IDF_TOOLS_PATH environment variable specifies the location where the tools are to be downloaded and installed. If not set, IDF_TOOLS_PATH defaults to HOME/.espressif on Linux and macOS, and %USER_PROFILE%\espressif on Windows.

Inside IDF_TOOLS_PATH, the scripts performing tools installation create the following directories and files:

- dist — where the archives of the tools are downloaded.
- tools — where the tools are extracted. The tools are extracted into subdirectories: tools/TOOL_NAME/VERSION/. This arrangement allows different versions of tools to be installed side by side.
- idf-env.json — user install options (targets, features) are stored in this file. Targets are selected chip targets for which tools are installed and kept up-to-date. Features determine the Python package set which should be installed. These options will be discussed later.
- python_env — not tools related; virtual Python environments are installed in the sub-directories.
- espidf.constraints.*.txt — one constraint file for each ESP-IDF release containing Python package version requirements.

GitHub Assets Mirror

Most of the tools downloaded by the tools installer are GitHub Release Assets, which are files attached to a software release on GitHub.

If GitHub downloads are inaccessible or slow to access, it’s possible to configure a GitHub assets mirror.
To use Espressif’s download server, set the environment variable `IDF_GITHUB_ASSETS` to `dl.espressif.com/github_assets`. When the install process is downloading a tool from `github.com`, the URL will be rewritten to use this server instead.

Any mirror server can be used provided the URL matches the `github.com` download URL format: the install process will replace `https://github.com` with `https://${IDF_GITHUB_ASSETS}` for any GitHub asset URL that it downloads.

**Note:** The Espressif download server doesn’t currently mirror everything from GitHub, it only mirrors files attached as Assets to some releases as well as source archives for some releases.

### `idf_tools.py` script

`tools/idf_tools.py` script bundled with ESP-IDF performs several functions:

- **install:** Download the tool into `${IDF_TOOLS_PATH}/dist` directory, extract it into `${IDF_TOOLS_PATH}/tools/TOOL_NAME/VERSION`. The `install` command accepts the list of tools to install, in `TOOL_NAME` or `TOOL_NAME@VERSION` format. If all is given, all the tools (required and optional ones) are installed. If no argument or required is given, only the required tools are installed.
- **download:** Similar to `install` but doesn’t extract the tools. An optional `--platform` argument may be used to download the tools for the specific platform.
- **export:** Lists the environment variables which need to be set to use the installed tools. For most of the tools, setting `PATH` environment variable is sufficient, but some tools require extra environment variables. The environment variables can be listed in either of `shell` or `key-value` formats, set by `--format` parameter:
  - `shell` produces output suitable for evaluation in the shell. For example,
    ```bash
    export PATH="/home/user/.espressif/tools/tool/v1.0.0/bin:$PATH"
    ```
on Linux and macOS, and
    ```bash
    set "PATH=C:\Users\user\.espressif\tools\v1.0.0\bin;%PATH%"
    ```
on Windows.

**Note:** Exporting environment variables in Powershell format is not supported at the moment. The `key-value` format may be used instead.

The output of this command may be used to update the environment variables, if the shell supports this. For example:

```bash
eval $(IDF_PATH/tools/idf_tools.py export)
```

- **key-value** produces output in `VARIABLE=VALUE` format, suitable for parsing by other scripts:

  ```bash
  PATH=/home/user/.espressif/tools/tool/v1.0.0:$PATH
  ```

  Note that the script consuming this output has to perform expansion of `$VAR` or `%VAR%` patterns found in the output.

- **list:** Lists the known versions of the tools, and indicates which ones are installed.
- **check:** For each tool, checks whether the tool is available in the system path and in `IDF_TOOLS_PATH`. 
• install-python-env: Create a Python virtual environment in the \${IDF_TOOLS_PATH}/python_env directory and install there the required Python packages. An optional --features argument allows one to specify a comma-separated list of features to be added or removed. Feature that begins with - will be removed and features with + or without any sign will be added. Example syntax for removing feature XY is --features=-XY and for adding --features=+XY or --features=XY. If both removing and adding options are provided with the same feature, no operation is performed. For each feature a requirements file must exist. For example, feature XY is a valid feature if \${IDF_PATH}/tools/requirements/requirements.XY.txt is an existing file with a list of Python packages to be installed. There is one mandatory core feature ensuring core functionality of ESP-IDF (build, flash, monitor, debug in console). There can be an arbitrary number of optional features. The selected list of features is stored in idf-env.json. The requirement files contain a list of the desired Python packages to be installed and espidf.constraints.*.txt downloaded from https://dl.espressif.com and stored in \${IDF_TOOLS_PATH} the package version requirements for a given ESP-IDF version. Although it is not recommended, the download and use of constraint files can be disabled with the --no-constraints argument or setting the IDF_PYTHON_CHECK_CONSTRAINTS environment variable to no.

• check-python-dependencies: Checks if all required Python packages are installed. Packages from \${IDF_PATH}/tools/requirements/requirements.*.txt files selected by the feature list of idf-env.json are checked with the package versions specified in the espidf.constraints.*.txt file. The constraint file is downloaded with install-python-env command. The use of constraints files can be disabled similarly to the install-python-env command.

• uninstall: Print and remove tools, that are currently not used by active ESP-IDF version.
  – --dry-run Print installed unused tools.
  – --remove-archives Additionally remove all older versions of previously downloaded installation packages.

Install scripts

Shell-specific user-facing scripts are provided in the root of ESP-IDF repository to facilitate tools installation. These are:

• install.bat for Windows Command Prompt
• install.ps1 for Powershell
• install.sh for Bash
• install.fish for Fish

Aside from downloading and installing the tools into IDF_TOOLS_PATH, these scripts prepare a Python virtual environment, and install the required packages into that environment.

These scripts accept optionally a comma separated list of chip targets and --enable-* arguments for enabling features. These arguments are passed to the idf_tools.py script which stores them in idf-env.json. Therefore, chip targets and features can be enabled incrementally.

Running the scripts without any optional arguments will install tools for all chip targets (by running idf_tools.py install --targets=all) and Python packages for core ESP-IDF functionality (by running idf_tools.py install-python-env --features=core).

Or for example, install.sh esp32 will install tools only for ESP32. See the Getting Started Guide for more examples.

install.sh --enable-XY will enable feature XY (by running idf_tools.py install-python-env --features=core,XY).

Export scripts

Since the installed tools are not permanently added into the user or system PATH environment variable, an extra step is required to use them in the command line. The following scripts modify the environment variables in the current shell to make the correct versions of the tools available:

• export.bat for Windows Command Prompt
• export.ps1 for Powershell
export.sh for Bash  
export.fish for Fish

备注：To modify the shell environment in Bash, `export.sh` must be “sourced”: `. ./export.sh` (note the leading dot and space). `export.sh` may be used with shells other than Bash (such as zsh). However in this case the IDF_PATH environment variable must be set before running the script. When used in Bash, the script will guess the IDF_PATH value from its own location.

In addition to calling `idf_tools.py`, these scripts list the directories which have been added to the PATH.

Other installation methods

Depending on the environment, more user-friendly wrappers for `idf_tools.py` are provided:

- **IDF Tools installer for Windows** can download and install the tools. Internally the installer uses `idf_tools.py`.
- **Eclipse Plugin** includes a menu item to set up the tools. Internally the plugin calls `idf_tools.py`.
- **VSCode Extension** for ESP-IDF includes an onboarding flow. This flow helps setting up the tools. Although the extension does not rely on `idf_tools.py`, the same installation method is used.

Custom installation

Although the methods above are recommended for ESP-IDF users, they are not a must for building ESP-IDF applications. ESP-IDF build system expects that all the necessary tools are installed somewhere, and made available in the PATH.

List of IDF Tools

**xtensa-esp-elf-gdb**  
GDB for Xtensa  
License: GPL-3.0-or-later  
More info: https://github.com/espressif/binutils-gdb
<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-amd64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-x86_64-linux-gnu.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-x86_64-linux-gnu.tar.gz</a></td>
<td>SHA256: d0356f2435ef05cccadac5d8f8c3e56c3d853f58ba1ed9f501acfe4f7</td>
</tr>
<tr>
<td>linux-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-aarch64-linux-gnu.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-aarch64-linux-gnu.tar.gz</a></td>
<td>SHA256: 76c9674cc4f4c5e7b8c94ca65bce53de4a44cbbcc972a9caee6fcd6a872c804c02</td>
</tr>
<tr>
<td>linux-armel</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-arm-linux-gnueabi.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-arm-linux-gnueabi.tar.gz</a></td>
<td>SHA256: 68118f73e9d2328d92a7a529d0e28d20fd6426036a0736a1147935614ce2</td>
</tr>
<tr>
<td>linux-armhf</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-arm-linux-gnueabihf.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-arm-linux-gnueabihf.tar.gz</a></td>
<td>SHA256: 72d75d9bba9a09d0696aa86628b2dd1851755216b1b315743189ea37228e572f</td>
</tr>
<tr>
<td>linux-i686</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-i586-linux-gnu.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-i586-linux-gnu.tar.gz</a></td>
<td>SHA256: 68118f73e9d2328d92a7a529d0e28d20fd6426036a0736a1147935614ce2</td>
</tr>
<tr>
<td>macos</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-x86_64-apple-darwin14.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-x86_64-apple-darwin14.tar.gz</a></td>
<td>SHA256: 417fc8d1b596b9481603d6987def1d6f6cebeb973f95394087334a7de855fa</td>
</tr>
<tr>
<td>win32</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-i686-w64-mingw32.zip">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-i686-w64-mingw32.zip</a></td>
<td>SHA256: 642b6a13c3b8f1d3e58a2c2f29469b769f8c1b101dab363d06101b02284b979</td>
</tr>
<tr>
<td>win64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/xensta-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip</a></td>
<td>SHA256: 2d958570ff6aa69ed32cbb076c0b303349a26b301a7e6a28be8d7ad39e9f1</td>
</tr>
</tbody>
</table>

**riscv32-elf-gdb**  
GDB for RISC-V  
License: GPL-3.0-or-later  
More info: [https://github.com/espressif/binutils-gdb](https://github.com/espressif/binutils-gdb)
<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-amd64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-linux-gnu.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-linux-gnu.tar.gz</a></td>
<td>f0cf0821ea7c8f2c63b14f2b69d6f12f4f8c266b29d02d5547b7d7cbbd0e11</td>
</tr>
<tr>
<td>linux-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-aarch64-linux-gnu.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-aarch64-linux-gnu.tar.gz</a></td>
<td>6812344dfb5c50a81d2d8354463516f0aa5f582e8aaf406cbaca8722b45fa94</td>
</tr>
<tr>
<td>linux-armel</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-arm-linux-gnueabi.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-arm-linux-gnueabi.tar.gz</a></td>
<td>b73042b8edf5a3fc80083cc3d000ef579f155d72a666c8ade1d48906d843e738</td>
</tr>
<tr>
<td>linux-armhf</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-arm-linux-gnueabihf.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-arm-linux-gnueabihf.tar.gz</a></td>
<td>9d5334042a169606d32ee454d359d3216a24df027e4b0830ab268a8999db2d</td>
</tr>
<tr>
<td>linux-i686</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-i586-linux-gnu.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-i586-linux-gnu.tar.gz</a></td>
<td>3f07a1bbdc87127a1af6bec6fbace4f8daca44755356f0692e9a5d4c84b6fd81d</td>
</tr>
<tr>
<td>macos</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-apple-darwin14.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-apple-darwin14.tar.gz</a></td>
<td>bb139229f9a4998c89a9c6b617d3ecb05b77e7b9a9a3a59c54969035f1b4007487</td>
</tr>
<tr>
<td>macos-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-aarch64-apple-darwin21.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-aarch64-apple-darwin21.tar.gz</a></td>
<td>f6513b57f7282454977f39a201f3f63444d4180e16b39765c629e01036286c0e6</td>
</tr>
<tr>
<td>win32</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip</a></td>
<td>8287fa891e8d932e283210048d635305791eda31504369418288d3e4753dd6</td>
</tr>
<tr>
<td>win64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip">https://github.com/espressif/binutils-gdb/releases/download/esp-gdb-v12.1_20221002/riscv32-esp-elf-gdb-12.1_20221002-x86_64-w64-mingw32.zip</a></td>
<td>9debabc1135df85868a9d945468f0480daa025f77ead6a55cc85142c4487abd</td>
</tr>
</tbody>
</table>

**xtensa-esp32-elf**  Toolchain for Xtensa (ESP32) based on GCC

License: GPL-3.0-with-GCC-exception

More info: [https://github.com/espressif/crosstool-NG](https://github.com/espressif/crosstool-NG)
<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-amd64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-amd64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-amd64.tar.xz</a></td>
<td>698d8407e18275d18fbeb71a6f6b68808a97904f0e39080422f86809a49df30</td>
</tr>
<tr>
<td>linux-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-arm64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-arm64.tar.xz</a></td>
<td>48ed01a6bf1e89ef6e1c3ebe4e00df6a6f67e53ae24979790464a4a3b64aa622</td>
</tr>
<tr>
<td>linux-armel</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-armel.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-armel.tar.xz</a></td>
<td>0e6131a9ab6c3da0a153e2e7509701283ccf21f9069368c35f53c8a086736f8</td>
</tr>
<tr>
<td>linux-armhf</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-armhf.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-armhf.tar.xz</a></td>
<td>74173665e228d8b1c988de0d743607a2f661e2bd24619c246e25da7a0f1f6bd</td>
</tr>
<tr>
<td>linux-i686</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-i686.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-linux-i686.tar.xz</a></td>
<td>d06511bb18057d72b555dc5b62b0686f19e9f8c7d7eae218b712e0907db2</td>
</tr>
<tr>
<td>macos</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-macos.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-macos.tar.xz</a></td>
<td>1c9d873e56469e3aabece4214b7200d36804a605d4f0991e539b1577415409bf</td>
</tr>
<tr>
<td>macos-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-macos-arm64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-macos-arm64.tar.xz</a></td>
<td>297249b0dc5307fd496c4d85d960b69824996c0450a8e92f8414a5fd32a7c3b</td>
</tr>
<tr>
<td>win32</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-win32.zip">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-win32.zip</a></td>
<td>858ee049d6d8e730edc30285c4adc1a9cdef077b591ed0b6f2bfa5e3564f53</td>
</tr>
<tr>
<td>win64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-win64.zip">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32-elf-gcc11_2_0-esp-2022r1-win64.zip</a></td>
<td>f469aff6a71113e3a145466d814184339e02248b158357766970646f5d2a3da7</td>
</tr>
</tbody>
</table>

**xtensa-esp32s2-elf**  Toolchain for Xtensa (ESP32-S2) based on GCC

License: GPL-3.0-with-GCC-exception

More info: https://github.com/espressif/crosstool-NG
<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
</tr>
</thead>
</table>
| linux-amd64      | required | https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s2-elf-gcc11_2_0-esp-2022r1-linux-amd64.tar.xz  
SHA256: 56e5913b6662b8eecc7d6b46780e668bc7e7cebef239e326a74f764c92a3cc841 |
| linux-arm64      | required | https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s2-elf-gcc11_2_0-esp-2022r1-linux-arm64.tar.xz  
SHA256: 2f0cc92d0279d6407ed9547250fbd0434f16606faa94460c52b74614a38a1e21 |
SHA256: 71974c4aaf3f637f6adaa28bbbf3a911db3385e0ab1544844513ec65185cc5 |
SHA256: 73e3be22c993f1112fcbb17631d82552a6b759f82f12cfcf78e669c7303d92b25 |
SHA256: 504efe97ce24561537bd442494b1046fc8f89cc43ae06ef11afa4652b7517201 |
SHA256: f53a4923490001727c5b6c3b8e1602b887783f0ed68e5defbb3c7712ada9631 |
| macos-arm64      | required | https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s2-elf-gcc11_2_0-esp-2022r1-macos-arm64.tar.xz  
SHA256: 3592e0fdbh2ca438c7360d93fd62ef0e05ead2fc8144ef344b6e1971d33287 |
SHA256: 96b87321043873a5aee39e591cdbeef453cb431d8392ac3fa2e68a48bc97 |
| win64            | required | https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s2-elf-gcc11_2_0-esp-2022r1-win64.zip  
SHA256: 9ab0387e08047916bbf7ff0d2eb974c710bcb2e042cb0437b44d93c9186f676 |

**xtensa-esp32s3-elf**  Toolchain for Xtensa (ESP32-S3) based on GCC  
License: GPL-3.0-with-GCC-exception  
More info: https://github.com/espressif/crosstool-NG
### Chapter 4. API

#### Plaform Required Download

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-amd64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-amd64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-amd64.tar.xz</a> SHA256: 5058b2e724166c3ca409ec2d53773750252de8bce50398b06c00352f9a8151f76e</td>
</tr>
<tr>
<td>linux-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-arm64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-arm64.tar.xz</a> SHA256: d2ce6f98a5018139a9f5af6eb8b08e968f1381a5b34547a185f4dec142b0fa44e</td>
</tr>
<tr>
<td>linux-armel</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-armel.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-armel.tar.xz</a> SHA256: 9944e67d95a5de9875670c5cd5cb0bb282ebac235a385f6d53069893fead9e</td>
</tr>
<tr>
<td>linux-armhf</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-armhf.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-armhf.tar.xz</a> SHA256: 9944e67d95a5de9875670c5cd5cb0bb282ebac235a385f6d53069893fead9e</td>
</tr>
<tr>
<td>linux-i686</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-i686.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-linux-i686.tar.xz</a> SHA256: 9944e67d95a5de9875670c5cd5cb0bb282ebac235a385f6d53069893fead9e</td>
</tr>
<tr>
<td>macos</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-macos.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-macos.tar.xz</a> SHA256: 2b46730adc6afd8115e0be9365050a87f9523617e5e58eece8585f1d2756c</td>
</tr>
<tr>
<td>macos-arm64</td>
<td>optional</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-macos-arm64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-macos-arm64.tar.xz</a> SHA256: c0a8836dd709605f8d68ea1fd6e8ae79b3f176274bfffdd8e79eeacd8f1f3ce1</td>
</tr>
<tr>
<td>macos-armel</td>
<td>optional</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-macos-armel.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-macos-armel.tar.xz</a> SHA256: 9944e67d95a5de9875670c5cd5cb0bb282ebac235a385f6d53069893fead9e</td>
</tr>
<tr>
<td>win64</td>
<td>optional</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-win64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/xtensa-esp32s3-elf-gcc11_2_0-esp-2022r1-win64.tar.xz</a> SHA256: 7213a0bf22607e9c70febaabde737822c2ae5e071ac53d6467e6031bb0b0b2bf</td>
</tr>
</tbody>
</table>

**esp-clang** Toolchain for all Espressif chips based on clang

License: Apache-2.0

More info: https://github.com/espressif/llvm-project
**riscv32-esp-elf** Toolchain for 32-bit RISC-V based on GCC

License: GPL-3.0-with-GCC-exception

More info: [https://github.com/espressif/crosstool-NG](https://github.com/espressif/crosstool-NG)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-amd64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-linux-amd64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-linux-amd64.tar.xz</a></td>
<td>SHA256: 52710f804df3a033a2b621c16e9a121023b42052819a51e35a2a164140bbf665</td>
</tr>
<tr>
<td>linux-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-linux-arm64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-linux-arm64.tar.xz</a></td>
<td>SHA256: 812a18f2ecdc3f72c1d098c4e8baa9688410999ce9d9ef95b1ea85f71e11013</td>
</tr>
<tr>
<td>linux-armhf</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-linux-armhf.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-linux-armhf.tar.xz</a></td>
<td>SHA256: 63f85a089fcd06939ed5e7e72ee5edca590aa47007e5e49c0a4c59ef1cb3a7b</td>
</tr>
<tr>
<td>macos-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-macos-arm64.tar.xz">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-macos-arm64.tar.xz</a></td>
<td>SHA256: 23d9a715d932a3af57fd7393b0789f88df0f70eda5f8b04289bab3de81be271bd6</td>
</tr>
<tr>
<td>win32</td>
<td>required</td>
<td><a href="https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-win32.zip">https://github.com/espressif/crosstool-NG/releases/download/esp-2022r1/riscv32-esp-elf-gcc11_2_0-esp-2022r1-win32.zip</a></td>
<td>SHA256: 3e677ef068d7f154d33b0d3788b5f985c5066d110028eac44e0f76b3ba4a92b</td>
</tr>
</tbody>
</table>

**esp32ulp-elf** Toolchain for ESP32 ULP coprocessor

License: GPL-3.0-or-later

More info: [https://github.com/espressif/binutils-gdb](https://github.com/espressif/binutils-gdb)
### Platform | Required | Download |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-amd64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-linux-amd64.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-linux-amd64.tar.gz</a> SHA256: b1f7801c3a16162e72393ebb772c0cbfe4d22d907be7c2c2dac168736e9195f6</td>
</tr>
<tr>
<td>linux-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-linux-arm64.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-linux-arm64.tar.gz</a> SHA256: d6671b31bab31b9b13aea25bb7d60f15484cb8f961ddf67a62867e5563ea5</td>
</tr>
<tr>
<td>linux-armel</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-linux-armel.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-linux-armel.tar.gz</a> SHA256: e107e7a9c9ed50d630b034f435a16a52db5a57388dc639a99c4c393c5e429711e9</td>
</tr>
<tr>
<td>macos</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-macos.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-macos.tar.gz</a> SHA256: 5a952087b621ced16af1e375feae1371a61cb51ab7e7b44c8ebf5afda2d573de</td>
</tr>
<tr>
<td>macos-arm64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-macos-arm64.tar.gz">https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-macos-arm64.tar.gz</a> SHA256: 73bd4a476ef92d4f4abe96519abbba40e5e31f368427469447b83cc7bb9b42</td>
</tr>
<tr>
<td>win32</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-win32.zip">https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-win32.zip</a> SHA256: 77344715ea7d7a79fd0b27653f880efaf3bccc1ac843f61492d8a0365d917f31</td>
</tr>
<tr>
<td>win64</td>
<td>required</td>
<td><a href="https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-win64.zip">https://github.com/espressif/binutils-gdb/releases/download/esp32ulp-elf-v2.35_20220830/esp32ulp-elf-v2.35_20220830-win64.zip</a> SHA256: 525e5b4e8299869a3fdddb51baad766125c104bd96952ae6460ad7e5b5a4e21</td>
</tr>
</tbody>
</table>

**cmake**  
CMake build system

On Linux and macOS, it is recommended to install CMake using the OS package manager. However, for convenience it is possible to install CMake using `idf_tools.py` along with the other tools.

License: BSD-3-Clause

More info: [https://github.com/Kitware/CMake](https://github.com/Kitware/CMake)
### Chapter 4. API

#### Platform Required Download

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
</tr>
</thead>
</table>
| linux-amd64      | optional | https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-linux-x86_64.tar.gz  
|                  |          | SHA256: 726f88e56985239114ebce9b059dc260b851a77ff97e4cc5573f4e42775a5c16f |
| linux-arm64      | optional | https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-linux-aarch64.tar.gz  
|                  |          | SHA256: 50c3b8e9d3a3cde850dd1ea143df9d1ae546bc5e74dc6d223eeefc1979189651 |
|                  |          | SHA256: 7dc787ef968dfef92491a4f191b8739f7086a496e0b811ca737b52481be0 |
|                  |          | SHA256: 7dc787ef968dfef92491a4f191b8739f7086a496e0b811ca737b52481be0 |
|                  |          | SHA256: 3e0cca74a56d9027dabb8545a5a26e42f8e8b33be1b655d6a724187a345145e4 |
|                  |          | SHA256: 3e0cca74a56d9027dabb8545a5a26e42f8e8b33be1b655d6a724187a345145e4 |
| win32            | required | https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-windows-x86_64.zip  
|                  |          | SHA256: b1ad8c2dbf0778e3efcc9f6d1cd4a962e5c1af40aadbdeebed35074bcff2e103c |
| win64            | required | https://github.com/Kitware/CMake/releases/download/v3.24.0/cmake-3.24.0-windows-x86_64.zip  
|                  |          | SHA256: b1ad8c2dbf0778e3efcc9f6d1cd4a962e5c1af40aadbdeebed35074bcff2e103c |

---

**openocd-esp32** OpenOCD for ESP32

License: GPL-2.0-only

More info: https://github.com/espressif/openocd-esp32

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
</tr>
</thead>
</table>
| linux-amd64      | required | https://github.com/espressif/openocd-esp32/releases/download/v0.11.0-esp32-20221026.tar.gz  
|                  |          | SHA256: ce63e9b1dfab606cc62da5dc2aca22ba7036c42f74761c787eb02674ace7db |
| linux-arm64      | required | https://github.com/espressif/openocd-esp32/releases/download/v0.11.0-esp32-20221026.tar.gz  
|                  |          | SHA256: 6ef76101cca1964be30f74f191eef34abb423e32930a383012b866c9b76135 |
| linux-armhf      | required | https://github.com/espressif/openocd-esp32/releases/download/v0.11.0-esp32-20221026.tar.gz  
|                  |          | SHA256: 6ef76101cca1964be30f74f191eef34abb423e32930a383012b866c9b76135 |
| macos            | required | https://github.com/espressif/openocd-esp32/releases/download/v0.11.0-esp32-20221026.tar.gz  
|                  |          | SHA256: 6ef76101cca1964be30f74f191eef34abb423e32930a383012b866c9b76135 |
| macos-arm64      | required | https://github.com/espressif/openocd-esp32/releases/download/v0.11.0-esp32-20221026.tar.gz  
|                  |          | SHA256: 6ef76101cca1964be30f74f191eef34abb423e32930a383012b866c9b76135 |
| win32            | required | https://github.com/espressif/openocd-esp32/releases/download/v0.11.0-esp32-20221026.tar.gz  
|                  |          | SHA256: 6ef76101cca1964be30f74f191eef34abb423e32930a383012b866c9b76135 |
| win64            | required | https://github.com/espressif/openocd-esp32/releases/download/v0.11.0-esp32-20221026.tar.gz  
|                  |          | SHA256: 6ef76101cca1964be30f74f191eef34abb423e32930a383012b866c9b76135 |
ninja  Ninja build system

On Linux and macOS, it is recommended to install ninja using the OS package manager. However, for convenience it is possible to install ninja using idf_tools.py along with the other tools.

License: Apache-2.0

More info: https://github.com/ninja-build/ninja

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-amd64</td>
<td>optional</td>
<td><a href="https://dl.espressif.com/dl/ninja-1.10.2-linux64.tar.gz">https://dl.espressif.com/dl/ninja-1.10.2-linux64.tar.gz</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA256: 32bb769de4d57a7ee0e292f3c7b553e7cc8ea096f17aa2b3ae60aa407c4033</td>
</tr>
<tr>
<td>macos</td>
<td>optional</td>
<td><a href="https://dl.espressif.com/dl/ninja-1.10.2-osx.tar.gz">https://dl.espressif.com/dl/ninja-1.10.2-osx.tar.gz</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA256: 847bb1ca4bc16d8dbfaaed3ebcb05549b8b86bc68c3d64c37583eb5738bb440f1</td>
</tr>
<tr>
<td>macos-arm64</td>
<td>optional</td>
<td><a href="https://dl.espressif.com/dl/ninja-1.10.2-osx.tar.gz">https://dl.espressif.com/dl/ninja-1.10.2-osx.tar.gz</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA256: 847bb1ca4bc16d8dbfaaed3ebcb05549b8b86bc68c3d64c37583eb5738bb440f1</td>
</tr>
<tr>
<td>win64</td>
<td>required</td>
<td><a href="https://dl.espressif.com/dl/ninja-1.10.2-win64.zip">https://dl.espressif.com/dl/ninja-1.10.2-win64.zip</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA256: bbdde850d2d72737c5764c927d1071cb1f1957dacbda4a130fa8547c12c695f</td>
</tr>
</tbody>
</table>

idf-exe  IDF wrapper tool for Windows

License: Apache-2.0

More info: https://github.com/espressif/idf_py_exe_tool

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>win32</td>
<td>required</td>
<td><a href="https://github.com/espressif/idf_py_exe_tool/releases/download/v1.0.3/idf-exe-v1.0.3.zip">https://github.com/espressif/idf_py_exe_tool/releases/download/v1.0.3/idf-exe-v1.0.3.zip</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA256: 7c81ef534c562354a5402abbb90a6eb1c8473a9f4a7b7a7f93ebbd23b4a2755</td>
</tr>
<tr>
<td>win64</td>
<td>required</td>
<td><a href="https://github.com/espressif/idf_py_exe_tool/releases/download/v1.0.3/idf-exe-v1.0.3.zip">https://github.com/espressif/idf_py_exe_tool/releases/download/v1.0.3/idf-exe-v1.0.3.zip</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA256: 7c81ef534c562354a5402abbb90a6eb1c8473a9f4a7b7a7f93ebbd23b4a2755</td>
</tr>
</tbody>
</table>

ccache  Ccache (compiler cache)

License: GPL-3.0-or-later

More info: https://github.com/ccache/ccache

<table>
<thead>
<tr>
<th>Platform</th>
<th>Required</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>win64</td>
<td>required</td>
<td><a href="https://github.com/ccache/ccache/releases/download/v4.6.2/ccache-4.6.2-windows-x86_64.zip">https://github.com/ccache/ccache/releases/download/v4.6.2/ccache-4.6.2-windows-x86_64.zip</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA256: bf230b09369f2e43a3410d6477a7d0b9308e29f89a3091881d22e2502604c5</td>
</tr>
</tbody>
</table>

dfu-util  dfu-util (Device Firmware Upgrade Utilities)

License: GPL-2.0-only

More info: http://dfu-util.sourceforge.net/
### 4.31 ESP32 中的单元测试

ESP-IDF 提供以下方法测试软件。

- 一种是基于目标的测试, 该测试使用运行在 esp32 上的中央单元测试应用程序。这些测试使用的是基于 Unity 的单元测试框架。通过把测试用例放在组件的 test 子目录，可以将其集成到 ESP-IDF 组件中。本文档主要介绍这种方法的目标测试方法。
- 另一种是基于 Linux 主机的单元测试, 其中所有硬件行为都通过 Mock 组件进行模拟。此测试方法目前仍在开发中，暂且只有一小部分 IDF 组件支持 Mock，具体请参考基于 Linux 主机的单元测试。

#### 4.31.1 添加常规测试用例

单元测试被添加在相应组件的 test 子目录中。测试用例写在 C 文件中，一个 C 文件可以包含多个测试用例。测试文件的名字要以 “test” 开头。

测试文件需要包含 unity.h 头文件，此外还需要包含待测试 C 模块需要的头文件。

测试用例需要通过 C 文件中特定的函数来添加。如下所示:

```c
TEST_CASE("test name", "[module name]"
{
   // 在这里添加测试用例
}
```

- 第一个参数是此测试的描述性名称。
- 第二个参数是用方括号括起来的标识符。标识符用来对相关测试或具有特定属性的测试进行分组。

**备注：** 没有必要在每个测试用例中使用 UNITY_BEGIN() 和 UNITY_END() 来声明主函数的区域, unity_platform.c 会自动调用 UNITY_BEGIN(), 然后运行测试用例, 最后调用 UNITY_END()。

test 子目录应包含组件 CMakeLists.txt，因为他们本身就是一种组件（即测试组件）。ESP-IDF 使用了 Unity 测试框架, 基于 unity 组件里。因此, 每个测试组件都需要通过 REQUIRES 参数将 unity 组件设为依赖项。通常, 组件需要手动指定待编译的源文件, 但是, 对于测试组件来说, 这个要求被放宽为仅建议将参数 SRC_DIRS 用于 idf_component_register。

总的来说, test 子目录下最小的 CMakeLists.txt 文件可能如下所示:

```cmake
idf_component_register(SRC_DIRS "."
   INCLUDE_DIRS "."
   REQUIRES unity)
```

更多关于如何在 Unity 下编写测试用例的信息，请查阅 [http://www.throwtheswitch.org/unity](http://www.throwtheswitch.org/unity)。
4.31.2 添加多设备测试用例

常规测试用例会在一个在试设备 (Device Under Test, DUT) 上执行。但是，由于要求互相通信的组件（比如 GPIO、SPI）需要与其他设备进行通信，因此不能使用常规测试用例进行测试。多设备测试用例包括写入多个测试函数，并在多个 DUT 运行测试。

以下是一个多设备测试用例：

```c
void gpio_master_test()
{
    gpio_config_t slave_config = {
        .pin_bit_mask = 1 << MASTER_GPIO_PIN,
        .mode = GPIO_MODE_INPUT,
    };
    gpio_config(&slave_config);
    unity_wait_for_signal("output high level");
    TEST_ASSERT(gpio_get_level(MASTER_GPIO_PIN) == 1);
}
void gpio_slave_test()
{
    gpio_config_t master_config = {
        .pin_bit_mask = 1 << SLAVE_GPIO_PIN,
        .mode = GPIO_MODE_OUTPUT,
    };
    gpio_config(&master_config);
    gpio_set_level(SLAVE_GPIO_PIN, 1);
    unity_send_signal("output high level");
}
TEST_CASE_MULTIPLE_DEVICES("gpio multiple devices test example", "[driver]", gpio_<master_test, gpio_slave_test>);
```

宏 TEST_CASE_MULTIPLE_DEVICES 用来声明多设备测试用例。

- 第一个参数指定测试用例的名字。
- 第二个参数是测试用例的描述。
- 从第三个参数开始，可以指定最多 5 个测试函数，每个函数都是单独运行在一个 DUT 上的测试入口点。

在不同的 DUT 上运行的测试用例需要相互之间进行同步。可以通过 unity_wait_for_signal 和 unity_send_signal 这两个函数使用 UART 进行同步操作。上例的场景中，slave 应该在 master 设置好 GPIO 电平时再读取 GPIO 电平，DUT 的 UART 终端会打印提示信息，并要求用户进行交互。

DUT1 (master) 终端:

```
Waiting for signal: [output high level]!
Please press "Enter" key once any board send this signal.
```

DUT2 (slave) 终端:

```
Send signal: [output high level]!
```

一旦 DUT2 发送了该信号，您需要在 DUT1 的终端按回车键，然后 DUT1 会从 unity_wait_for_signal 函数中解除阻塞，并开始更改 GPIO 的电平。

4.31.3 添加多阶段测试用例

常规的测试用例无需重启就会结束（或者仅需要检查是否发生了重启），可有些时候我们想在某些特定类型的重启事件后运行指定的测试代码。例如，在深度睡眠唤醒后检查复位的原因是否正确。首先我们需要触发深度睡眠复位事件，然后检查复位的原因。为了实现这一点，可以通过定义多阶段测试用例来将这些测试函数组合在一起。
多阶段测试用例向用户呈现了一组测试函数，它需要用户进行交互（选择用例并选择不同的阶段）来运行。

### 4.31.4 应用于不同芯片的单元测试

某些测试（尤其与硬件相关的）不支持在所有的芯片上执行。请参照本节，让您的单元测试只在其中一部分芯片上执行。

1. 使用宏！(TEMPORARY_DISABLED_FOR_TARGETS()) 包装您的测试代码，并将其放于原始的测试文件中，或将代码分成按功能分组的文件。但请确保所有这些文件都会由编译器处理。例:

```c
#if !TEMPORARY_DISABLED_FOR_TARGETS(ESP32, ESP8266)
TEST_CASE("a test that is not ready for esp32 and esp8266 yet", "]")
{
}
#endif //TEMPORARY_DISABLED_FOR_TARGETS(ESP32, ESP8266)
```

如果您需要将其中某个测试在特定芯片上编译，只需要修改禁止的芯片列表。推荐使用一些能在 soc_caps.h 中被清楚描述的通用概念来禁止某些单元测试。如果您已经进行上述操作，但一些测试在芯片中的调整未通过，请同时使用上述两种方法，当测试完成后再移除！(TEMPORARY_DISABLED_FOR_TARGETS())。例:

```c
#if SOC_SDIO_SLAVE_SUPPORTED
#if !TEMPORARY_DISABLED_FOR_TARGETS(ESP64)
TEST_CASE("a sdio slave tests that is not ready for esp64 yet", "][sdio_slave]"
{
    //available for esp32 now, and will be available for esp64 in the future
}
#endif //TEMPORARY_DISABLED_FOR_TARGETS(ESP64)
#endif //SOC_SDIO_SLAVE_SUPPORTED
```

2. 对于某些您确定不会支持的测试 (例如，芯片根本没有该外设)，使用 DISABLED_FORTARGETS 来禁止该测试；对于其他只是临时性需要关闭的（例如，没有 runner 资源等）使用 TEMPORARY_DISABLED_FORTARGETS 来暂时关闭该测试。

一些禁用目标芯片测试用例的旧方法，由于它们具有明显的缺陷，已经被废弃，请勿继续使用：

- 请勿将测试代码放在 test/target 目录下并用 CMakeLists.txt 来选择其中一个进行编译。这是因为测试代码比实现代码更容易被复用。如果您将一些代码放在 test/esp32 目录下，您将无法执行它，一旦您需要在新的芯片（比如 esp32s3）中启用该测试，这种结构将变得难以保持代码整洁。
- 请勿继续使用 CONFIG_IDF_TARGET_xxx 宏来禁用测试。这种方法会导致被禁用的测试被忽略，功能可能会被忽略。并且，相比于自定义的 #if CONFIG_IDF_TARGET xxx，黑名单式的 #if !disabled 不会导致在新芯片上插入这些测试被自动禁用。但对于测试实现，仍然可以使用 #if CONFIG_IDF_TARGET xxx 给不同芯片版本选择实现代码。在测试项目中使用这种功能时，请自行判断如下:
  - 测试项目：那些会在一些芯片上执行，而在另外一些芯片上跳过的项目，例如:
有三个测试项目 SD 1-bit，SD 4-bit 和 SDSPI。对于不支持 SD Host 外设的 ESP32-S2 芯片，只有 SDSPI 一个项目需要被执行。

测试实现：一些始终会发生的代码，但采取的实现方式不同。例如：ESP3260 的 SDIO_PKT_LEN 寄存器。如果在测试过程中需要从 slave 设备的数据长度，您可以不同方式读取的 #if CONFIG_IDF_TARGET_宏来保护不同的实现代码。但请注意避免使用 #else 宏。这样当新芯片被引入时，测试就会在编译阶段失败，提示维护者去显示选择一个正确的测试实现。

### 4.31.5 编辑单元测试程序

按照 esp-idf 顶层目录的 README 文件中的说明进行操作，确保 IDF_PATH 环境变量已经被设置指向了 esp-idf 的顶层目录。

切换到 tools/unit-test-app 目录下进行配置和编译：

- `idf.py menuconfig` 配置单元测试程序。
- `idf.py -T all build` 编译单元测试程序，测试每个组件 test 子目录下的用例。
- `idf.py -T "xxx yyy" build` 编译单元测试程序，对已空格分隔的特定组件进行测试（如 `idf.py -T heap build` 仅对 heap 组件目录下的单元测试程序进行编译）。
- `idf.py -T all -E "xxx yyy" build` 编译单元测试程序，测试除指定组件之外的所有组件（例如 `idf.py -T all -E "ulp mbedtls" build` 编译所有的单元测试，不包括 ulp 和 mbedtls 组件）。

**备注：** 由于 Windows 命令提示符有限制，需使用以下语法来编译多个组件的单元测试程序：`idf.py -T xxx -T yyy build` 或者在 PowerShell 中使用 `idf.py -T "xxx yyy" build`。在 Windows 命令提示符下使用 `idf.py -T "ssd1306 hts221" build`。

当编译完成时，它会打印出烧写芯片的指令。您只需要运行 `idf.py flash` 即可烧写所有编译输出的文件。

您还可以运行 `idf.py -T all flash` 或者 `idf.py -T xxx flash` 来编译并烧写，所有需要的文件都会在烧写之前自动重新编译。

使用 `menuconfig` 可以设置烧写测试程序所使用的串口。更多信息，见 `tools/unit-test-app/README.md`。

### 4.31.6 运行单元测试

烧写完成后重启 ESP32，它将启动单元测试程序。

当单元测试应用程序空闲时，输入回车键，它会打印出测试菜单，其中包含所有的测试项目：

Here's the test menu, pick your combo:

1. "esp_ota_begin() verifies arguments" [ota]
2. "esp_ota_get_next_update_partition logic" [ota]
3. "Verify bootloader image in flash" [bootloader_support]
4. "Verify unit test app image" [bootloader_support]
5. "can use new and delete" [cxx]
6. "can call virtual functions" [cxx]
7. "can use static initializers for non-POD types" [cxx]
8. "can use std::vector" [cxx]
9. "static initialization guards work as expected" [cxx]
10. "global initializers run in the correct order" [cxx]
11. "before scheduler has started, static initializers work correctly" [cxx]
12. "adc2 work with wifi" [adc]
13. "gpio master/slave test example" [ignore][misc][test_env=UT_T2_1][multi_device]
   1. "gpio_master_test"
   2. "gpio_slave_test"
14. "SPI Master clockdiv calculation routines" [spi]
常规测试用例会打印用例名字和描述，主从测试用例还会打印子菜单（已注册的测试函数的名字）。

可以输入以下任意一项来运行测试用例：
- 引号中写入测试用例的名字，运行单个测试用例。
- 测试用例的序号，运行单个测试用例。
- 方括号中的模块名字，运行指定模块所有的测试用例。
- 星号，运行所有测试用例。

`[multi_device]` 和 `[multi_stage]`` 标签告诉测试运行者该用例是多设备测试还是多阶段测试。这些标签由 `TEST_CASE_MULTIPLE_DEVICES` 和 `TEST_CASE_MULTIPLE_STAGES` 宏自动生成。

一旦选择了多设备测试用例，它会打印一个子菜单：

```plaintext
Running gpio master/slave test example...
gpio master/slave test example
   (1)  "gpio_master_test"
   (2)  "gpio_slave_test"
```

您需要输入数字以选择在 DUT 上运行的测试。

与多设备测试用例相似，多阶段测试用例也会打印子菜单：

```plaintext
Running reset reason check for deepsleep...
reset reason check for deepsleep
   (1)  "trigger_deepsleep"
   (2)  "check_deepsleep_reset_reason"
```

第一次执行此用例时，输入 1 运行第一阶段（触发深度睡眠）。在重启 DUT 并再次选择运行此用例后，输入 2 运行第二阶段。只有在最后一个阶段通过并且之前的阶段都成功触发了复位的情况下，该测试才算通过。

### 4.31.7 带缓冲补偿定时器的定时期代码

存储在外部存储器（如 SPI Flash 和 SPI RAM）中的指令和数据是通过 CPU 的统一指令和数据缓存来访问的。当代码或数据在缓存中时，访问速度会非常快（即缓存命中）。然而，如果指令或数据不在缓存中，则需要从外部存储器中获取（即缓存缺失）。访问外部存储器的速度明显较慢，因为 CPU 在等待从外部存储器获取指令或数据时会陷入停滞，从而导致整体代码执行速度会依据缓存命中或缓存缺失的次数而变化。

在不同的编译中，代码和数据的位置可能会有所不同，一些可能会更有利于缓存访问（即最大限度地减少缓存缺失）。理论上，这会影响执行速度，但这些因素通常无关紧要，因为它们的影响会在设备的运行过程中“平均化”。

然而，高速缓存对执行速度的影响可能与基准测试场景（尤其是微基准测试）有关。每次运行时间和构建时的测量时间可能会有所差异。减少差异的方法之一是将代码和数据分别放在指令或数据 RAM (IRAM/DRAM) 中。CPU 可以直接访问 IRAM 和 DRAM，从而消除了高速缓存的影响。然而，由于 IRAM 和 DRAM 容量有限，该方法并不总是可行。

缓存补偿定时器是上述方法的替代方法，该计时器使用处理器的内部事件计数器来确定在发生高速缓存未命中时等待代码/数据所花费的时间，然后从记录的实时时间中减去该时间。
缓存补偿定时器的限制之一是基准功能必须固定在一个内核上。这是由于每个内核都有自己的事件计数器，这些事件计数器彼此独立。例如，如果在一个内核上调用 ccomp_timer_start，使调度器进入睡眠状态，唤醒并再在另一个内核上重新调度，那么对应的 ccomp_timer_stop 将无效。

### 4.31.8 Mocks

**备注**：目前，只有一些特定的组件在 Linux 主机上运行时才能 Mock。未来我们计划，无论是在 Linux 主机上运行还是在目标芯片 ESP32 上运行，IDF 所有重要的组件都可以实现 Mock。

嵌入式系统中单元测试的最大问题之一是对硬件依赖性极强。直接在 ESP32 上运行单元测试对于上层组件来说存在极大的困难，原因如下：

- 受下层组件和/或硬件设置的影响，测试可复性降低。
- 由于下层组件和/或硬件设置的限制，测试边缘案例的难度提高。
- 由于数量庞大的依赖关系影响了行为，识别根本原因的难度提高。

当测试一个特定的组件（即被测组件）时，通过软件进行 Mock 能让所有被测组件的依赖在软件中被完全替换（即 Mock）。为了实现此功能，ESP-IDF 集成了 CMock 的 Mock 框架作为组件，通过在 ESP-IDF 的构建系统中添加一些 CMake 函数，可以方便地 Mock 整个（或部分）IDF 组件。

理想情况下，被测组件所依赖的所有组件都应该被 Mock，从而让测试环境完全控制与被测组件之间的所有交互。然而，如果 Mock 所有的组件过于复杂或冗长（例如需要模拟过多的函数调用），以下做法可能会有帮助：

- 在测试代码中包含更多“真正”（非模拟）代码。这样做可能有效，但同时也会增加对“真正”代码行为的依赖。此外，一旦测试失败，很难判断失败原因是因为实际测试代码还是“真正”地 IDF 代码。
- 重新评估被测代码的设计，尝试将被测代码划分为更易于管理的组件来减少其依赖性。这可能看起来很麻烦，但众所周知，单元测试经常暴露软件设计的弱点。修复设计上的弱点不仅在短期内有助于进行单元测试，而且还有助于长期的代码维护。

请参考 cmock/CMock/docs/CMock_Summary.md 了解 CMock 工作原理以及如何创建和使用 Mock。

**要求**

目前 Mock 只支持基于 Linux 主机的单元测试。生成 Mock 需要满足如下要求：

- Installed IDF including all IDF requirements
- CMock requirements (Ruby)
- System package requirements (libbsd, libbsd-dev)

**对组件进行 Mock**

要创建组件的 Mock 版本（也称为“组件模拟”），需要以特定方式覆盖组件。覆盖组件时需要创建一个与原始组件名称完全相同的组件，然后让构建系统先发现原始组件再发现这个具有相同名称的新组件。具体可参考同名组件。
在组件模拟中需要指定如下部分：

- 头文件，头文件中提供了需要生成模拟的函数
- 上述头文件的路径
- 模拟组件的依赖（如果头文件中包含有其他组件的文件，那么这点非常必要）

以上这些部分都需要使用 IDF 构建系统函数 `idf_component_mock` 指定。您可以使用 IDF 构建系统函数 `idf_component_get_property`，并加上标签 `COMPONENT_OVERRIDEN_DIR` 访问原始组件的组件目录，然后使用 `idf_component_mock` 注册模拟组件。

组件模拟还需要一个单独的 mock 目录，里面包含一个 mock_config.yaml 文件用于配置 CMock。以下是一份简单的 mock_config.yaml 文件：

```yaml
:cmock:
:plugins:
  - expect
  - expect_any_args
```

更多关于 CMock yaml 类型配置文件的详细信息，请查看 cmock/CMock/docs/CMock_Summary.md。

请注意，组件模拟不一定要对原始组件进行整体模拟。只要组件模拟满足测试项目的依赖以及其他代码对原始组件的依赖，部分模拟就足够了。事实上，IDF 中 `tools/mocks` 中的大多数组件模拟都只是部分地模拟了原始组件。

可在 IDF 目录的 `tools/mocks` 下找到组件模拟的示例。有关如何覆盖 IDF 组件，可查看同名组件。

### 修改单元测试文件

单元测试需要通知 `cmake` 构建系统对依赖的组件进行模拟（即用模拟组件来覆盖原始组件）。这可以通过将组件模拟放到项目的 components 目录，或者在项目的根目录 `CMakeLists.txt` 文件中使用以下代码来添加模拟组件的目录来实现：

```cmake
list(APPEND EXTRA_COMPONENT_DIRS "<mock_component_dir>")
```

这两种方法都会让组件模拟覆盖 ESP-IDF 中的现有组件。如果使用的是 IDF 提供的组件模拟，则第二个方法更加方便。

您可参考 esp_event 基于主机的单元测试及其 esp_event/host_test/esp_event_unit_test/CMakeLists.txt 作为组件模拟的示例。

### 4.32 Unit Testing on Linux

**备注：** Host testing with IDF is experimental for now. We try our best to keep interfaces stable but can’t guarantee it for now. Feedback via github or the forum on esp32.com is highly welcome, though and may influence the future design of the host-based tests.

This article provides an overview of unit tests with IDF on Linux. For using unit tests on the target, please refer to target based unit testing.
4.32.1 Embedded Software Tests

Embedded software tests are challenging due to the following factors:

- Difficulties running tests efficiently.
- Lack of many operating system abstractions when interfacing with hardware, making it difficult to isolate code under test.

To solve these two problems, Linux host-based tests with CMock are introduced. Linux host-based tests are more efficient than unit tests on the target since they:

- Compile the necessary code only
- Don’t need time to upload to a target
- Run much faster on a host-computer, compared to an ESP

Using the CMock framework also solves the problem of hardware dependencies. Through mocking, hardware details are emulated and specified at run time, but only if necessary.

Of course, using code on the host and using mocks does not fully represent the target device. Thus, two kinds of tests are recommended:

1. Unit tests which test program logic on a Linux machine, isolated through mocks.
2. System/Integration tests which test the interaction of components and the whole system. They run on the target, where irrelevant components and code may as well be emulated via mocks.

This documentation is about the first kind of tests. Refer to target based unit testing for more information on target tests (the second kind of tests).

4.32.2 IDF Unit Tests on Linux Host

The current focus of the Linux host tests is on creating isolated unit tests of components, while mocking the component’s dependencies with CMock.

A complete implementation of IDF to run on Linux does not exist currently.

Examples for running IDF-built code on Linux host include (non-exhaustive list):

- unit test for the NVS Page class.
- unit test for esp_event.
- unit test for mqtt.

Inside the component which should be tested, there is a separate directory host_test, besides the “traditional” test directory or the test_apps directory. It has one or more subdirectories:

- host_test/
  - fixtures/
    contains test fixtures (structs/functions to do test case setup and tear-down).
    If there are no fixtures, this can be omitted.
  - <test_name>/
    IDF applications which run the tests
  - <test_name2>/
    Further tests are possible.

The IDF applications inside host_test set the mocking configuration as described in the IDF unit test documentation.

The NVS page unit test provides some illustration of how to control the mocks.

Requirements

- Installed IDF including all IDF requirements
• CMock requirements (Ruby)
• System package requirements (libbsd, libbsd-dev)

The host tests have been tested on Ubuntu 20.04 with GCC version 9 and 10.

4.33 Wi-Fi 驱动程序

4.33.1 ESP32 Wi-Fi 功能列表

ESP32 支持以下 Wi-Fi 功能:
• 支持 4 个虚拟接口，即 STA、AP、Sniffer 和 reserved。
• 支持仅 station 模式、仅 AP 模式、station/AP 共存模式
• 支持使用 IEEE 802.11b、IEEE 802.11g、IEEE 802.11n 和 AP 配置协议模式
• 支持 WPA/WPA2/WPA3/WPA2-企业版/WPA3-企业版/WAPI/WPS 和 DPP
• 支持 AMSDU、AMPDU、HT40、QoS 以及其它主要功能
• 支持 Modem-sleep
• 支持乐鑫专属协议，可实现 1 km 数据通信量
• 空中数据传输最高可达 20 MBit/s TCP 吞吐量和 30 MBit/s UDP 吞吐量
• 支持 Sniffer
• 支持快速扫描和全信道扫描
• 支持多个天线
• 支持获取信道状态信息

4.33.2 如何编写 Wi-Fi 应用程序

准备工作

一般来说，要编写自己的 Wi-Fi 应用程序，最高效的方式是先选择一个相似的应用程序示例，然后将其中可用的部分移植到自己的项目中。如果您希望编写一个强健的 Wi-Fi 应用程序，强烈建议您在开始之前先阅读本文。非强制要求，请依个人情况而定。

本文将补充说明 Wi-Fi API 和 Wi-Fi 示例的相关信息，重点描述使用 Wi-Fi API 的原则、当前 Wi-Fi API 实现的限制以及使用 Wi-Fi 时的常见错误。同时，本文还介绍了 Wi-Fi 驱动程序的一些设计细节。建议您选择一个示例 example 进行参考。

设置 Wi-Fi 编译时选项

请参阅 Wi-Fi menuconfig。

Wi-Fi 初始化

请参阅 ESP32 Wi-Fi station 一般情况、ESP32 Wi-Fi AP 一般情况。

启动/连接 Wi-Fi

请参阅 ESP32 Wi-Fi station 一般情况、ESP32 Wi-Fi AP 一般情况。
第4章 API

事件处理

通常，在理想环境下编写代码难度并不大，如 WIFI_EVENT_STA_START、WIFI_EVENT_STA_CONNECTED 中所述。难度在于如何在现实的困难环境下编写代码，如 WIFI_EVENT_STA_DISCONNECTED 中所述。能否在后者情况下完美地解决各类事件冲突，是编写一个强健的 Wi-Fi 应用程序的根本。请参阅 ESP32 Wi-Fi 事件描述，ESP32 Wi-Fi Station 一般情况，ESP32 Wi-Fi AP 一般情况。另可参阅 ESP-IDF 中的事件处理概述。

编写错误恢复程序

除了在能在比较差的环境下工作，错误恢复能力也对一个强健的 Wi-Fi 应用程序至关重要。请参阅 ESP32 Wi-Fi API 错误代码。

4.33.3 ESP32 Wi-Fi API 错误代码

所有 ESP32 Wi-Fi API 都有定义好的返回值，即错误代码。这些错误代码可分类为:

- 无错误，例如：返回值 ESP_OK 代表 API 成功返回
- 可恢复错误，例如：ESP_ERR_NO_MEM
- 不可恢复的非关键性错误
- 不可恢复的关键性错误

一个错误是否为关键性取决于其 API 和应用场景，并且由 API 用户定义。

要使用 Wi-Fi API 编写一个强健的应用程序，根本原则便是要时刻检查错误代码并编写相应的错误处理代码。一般来说，错误处理代码可用于解决:

- 可恢复错误，您可以编写一个可恢复错误处理代码解决该类错误。例如，当 esp_wifi_start() 返回 ESP_ERR_NO_MEM 时，调用 esp_wifi_set_mode() 以获取几秒的重试时间。
- 不可恢复非关键性错误，打印错误代码可以帮助您更好地处理该类错误。
- 不可恢复关键性错误，可使用 “assert” 语句处理该类错误。例如，如果 esp_wifi_set_mode() 返回 ESP_ERR_WIFI_NOT_INIT，该值意为 esp_wifi_init() 未成功初始化 Wi-Fi 驱动程序。您可以在应用程序开发阶段非常快速地检测到此类错误。

在 esp_err.h 中，ESP_ERROR_CHECK 负责检查返回值。这是一个较为常见的错误处理代码，可在应用程序开发阶段作为默认的错误处理代码。但是，我们强烈建议 API 的使用者编写自己的错误处理代码。

4.33.4 初始化 ESP32 Wi-Fi API 参数

初始化 API 的结构参数时，应遵循以下两种方式之一:

- 设置该参数的所有字段
- 先使用 get API 获取当前配置，然后只设置特定于应用程序的字段

初始化或获取整个结构这一步至关重要，因为在大多数情况下，返回值 0 意味着程序使用了默认值。未来，我们将会在该结构中加入更多字段，并将这些字段初始化为 0，确保即便 IDF 版本升级后您的应用程序依然能够正常运行。

4.33.5 ESP32 Wi-Fi 编程模型

ESP32 Wi-Fi 编程模型如下图所示：

Wi-Fi 驱动程序可以看作是一个无法感知上层代码（如 TCP/IP 堆栈，应用程序任务，事件任务等）的黑匣子。通常，应用程序任务（使用 Wi-Fi 驱动程序 API 来初始化 Wi-Fi，并在必要时处理 Wi-Fi 事件。然后，Wi-Fi 驱动程序接收并处理 API 数据，并在应用程序中插入事件。

Wi-Fi 事件处理是在 esp_event 库的基础上进行的。Wi-Fi 驱动程序将事件发送至默认事件循环，应用程序便可以使用 esp_event_handler_register() 中的回调函数处理这些事件。除此之外，esp_netif 接
图 54: Wi-Fi 编程模型

也负责处理 Wi-Fi 事件，并产生一系列默认行为。例如，当 Wi-Fi station 连接至一个 AP 时，esp_netif 将自动开启 DHCP 客户端服务（系统默认）。

4.33.6 ESP32 Wi-Fi 事件描述

WIFI_EVENT_WIFI_READY

Wi-Fi 驱动程序永远不会生成此事件，因此，应用程序的事件回调函数可忽略此事件。在未来的版本中，此事件可能会被移除。

WIFI_EVENT_SCAN_DONE

扫描完成事件，由 esp_wifi_scan_start() 函数触发，将在以下情况下产生：

- 扫描已完成，例如：Wi-Fi 已成功找到目标 AP 或已扫描所有信道。
- 当前扫描因函数 esp_wifi_scan_stop() 而终止。
- 在当前扫描完成之前调用了函数 esp_wifi_scan_start()。此时，新的扫描将覆盖当前扫描过程，并生成一个扫描完成事件。

以下情况下将不会产生扫描完成事件：

- 当前扫描被阻止。
- 当前扫描是由函数 esp_wifi_connect() 触发的。

接收此事件后，事件任务暂不做任何响应。首先，应用程序的事件回调函数需调用 esp_wifi_scan_get_ap_num() 和 esp_wifi_scan_get_ap_records() 获取已扫描的 AP 列表，然后触发 Wi-Fi 驱动程序释放扫描过程中占用的内存空间（切记该步骤）。更多详细信息，请参阅 ESP32 Wi-Fi 扫描。

WIFI_EVENT_STA_START

如果调用函数 esp_wifi_start() 后接收到返回值 ESP_OK，且当前 Wi-Fi 处于 station 或 station/AP 共存模式，则将产生此事件。接收到此事件后，事件任务将初始化 LwIP 网络接口 (netif)。通常，应用程序的事件回调函数需调用 esp_wifi_connect() 来连接已配置的 AP。
Chapter 4. API 指南

WIFI_EVENT_STA_STOP

如果调用函数 esp_wifi_stop() 后接收到返回值 ESP_OK，且当前 Wi-Fi 处于 station 或 station/AP 共存模式，则将产生此事件。接收到此事件后，事件任务将进行释放 station IP 地址，终止 DHCP 客户端服务、移除 TCP/UDP 相关连接并清除 LwIP station netif 等动作。此时，应用程序的事件回调函数通常不需做任何响应。

WIFI_EVENT_STA_CONNECTED

如果调用函数 esp_wifi_connect() 后接收到返回值 ESP_OK，且 station 已成功连接目标 AP，则将产生此连接事件。接收到此事件后，事件任务将启动 DHCP 客户端服务并开始获取 IP 地址。此时，Wi-Fi 驱动程序已准备就绪，可发送和接收数据。如果您的应用程序不依赖于 LwIP（即 IP 地址），则此刻便可以开始应用程序开发工作。但是，如果您的应用程序需基于 LwIP 进行，则还需等待 got ip 事件发生后才可开始。

WIFI_EVENT_STA_DISCONNECTED

此事件将在以下情况下产生:

- 调用了函数 esp_wifi_disconnect() 或 esp_wifi_stop()，且 Wi-Fi station 已成功连接至 AP。
- 调用了函数 esp_wifi_connect()，但 Wi-Fi 驱动程序因为某些原因未能成功连接至 AP，例如:未扫描到目标 AP、验证超时等。或存在多个 SSID 相同的 AP，station 无法连接所有已找到的 AP，也将产生该事件。
- Wi-Fi 连接因为某些原因而中断，例如: station 连续多次丢失 N beacon、AP 踢掉 station、AP 认证模式改变等。

接收到此事件后，事件任务的默认动作作为:

- 关闭 station 的 LwIP netif。
- 通知 LwIP 任务清除导致所有套接字状态错误的 UDP/TCP 连接。针对基于套接字编写的应用程序，其回调函数可以在接收到此事件时（如有必要）关闭并重新创建所有套接字。

应用程序处理此事件最常用的方法为: 调用函数 esp_wifi_disconnect() 重新连接 Wi-Fi，但是，如果此次事件是由函数 esp_wifi_disconnect() 引发的，则应用程序不应调用 esp_wifi_connect() 来重新上线。应用程序须明确区分此事件的引发原因，因某些情况下使用其它更好的方式进行重新连接。请参阅 Wi-Fi 重新连接 和连接 Wi-Fi 时扫描。

需要注意的另一点是: 接收到此事件后，LwIP 的默认动作是终止所有 TCP 套接字连接。大多数情况下，该动作不会造成影响。但对某些特殊应用程序可能例外。例如:

- 应用程序创建一个了 TCP 连接，以维护每 60 秒发送一次的应用程序级、保持活动状态的数据。
- 由于某些原因，Wi-Fi 连接被中断并引发了 WIFI_EVENT_STA_DISCONNECTED 事件。根据当前实现，此时所有 TCP 连接都将被移除，且保持活动的套接字将处于错误的状态中。但是，由于应用程序设计者认为网络层不应考虑这个 Wi-Fi 层的错误，因此应用程序不会关闭套接字。
- 5 秒后，因为在应用程序的事件回调函数中调用了 esp_wifi_connect()，Wi-Fi 连接恢复，同时，station 连接至同一个 AP 并获得与之前相同的 IPV4 地址。
- 60 秒后，当应用程序发送具有保持活动状态的套接字的数据时，套接字将返回错误，应用程序将关闭套接字并在必要时重新创建。

在上述场景中，理想状态下应用层套接字和网络层将不会受到影响，因为在此过程中 Wi-Fi 连接只是短暂地断开然后快速恢复。应用程序可通过 LwIP menuconfig 启动“IP 改变时保持 TCP 连接”的功能。

IP_EVENT_STA_GOT_IP

当 DHCP 客户端成功从 DHCP 服务器获得 IPV4 地址或 IPV4 地址发生改变时，将引发此事件。此事件意味着应用程序一切就绪，可以开始任务（如：创建套接字）。

IPV4 地址可能由于以下原因而发生改变:

- DHCP 客户端无法重新获取/绑定 IPV4 地址，且 station 的 IPV4 重置为 0。
Chapter 4. API

- DHCP 客户端重新绑定新地址。
- 静态配置的 IPv4 地址已发生改变。

函数 ip_event_got_ip 中的字段 ip_change 说明了 IPv4 地址是否发生改变。

套接字的状态是基于 IPv4 地址的，这意味着，如果 IPv4 地址发生改变，则所有与此 IPv4 相关的套接字都将变为异常。接收到此事件后，应用程序需关闭所有套接字，并在 IPv4 变为有效地址时重新创建应用程序。

**IP_EVENT_GOT_IP6**

当 IPv6 SLAAC 支持自动为 ESP32 配置一个地址，或 ESP32 地址发生改变时，将引发此事件。此事件意味着应用程序必须重新启动，可以开始任务（如：创建套接字）。

**IP_EVENT_STA_LOST_IP**

当 IPv4 地址失效时，将引发此事件。

此事件不会在 Wi-Fi 断连后立刻出现。Wi-Fi 连接断开后，首先将启动一个 IPv4 地址丢失计时器。如果 station 在该计时器超时之前成功获取了 IPv4 地址，则不会发生此事件。否则，此事件将在计时器超时后发生。

一般来说，应用程序可忽略此事件。这只是一个调试事件，主要使应用程序获知 IPv4 地址已丢失。

**WIFI_EVENT_AP_START**

与 WIFI_EVENT_STA_START 事件相似。

**WIFI_EVENT_AP_STOP**

与 WIFI_EVENT_STA_STOP 事件相似。

**WIFI_EVENT_AP_STACONNECTED**

每当有一个 station 成功连接 ESP32 AP 时，将引发此事件。接收到此事件后，事件任务将不做任何响应。应用程序的回调函数也可忽略这一事件。但是，您可以在此时进行一些操作，例如：获取已连接 station 的信息等。

**WIFI_EVENT_AP_STATADISCONNECTED**

此事件将在以下情况下发生：

- 应用程序通过调用函数 esp_wifi_disconnect() 或 esp_wifi_deauth_sta() 手动断开 station 连接。
- Wi-Fi 驱动程序由于某些原因断开 station 连接，例如：AP 在过去 5 分钟（可改通过函数 esp_wifi_set_inactive_time() 修改该时间）内未接收到任何数据包等。
- station 断开与 AP 之间的连接。

发生此事件时，事件任务将不做任何响应，但应用程序的事件回调函数需执行一些操作，例如：关闭与此 station 相关的套接字等。

**WIFI_EVENT_AP_PROBEREQRECVED**

默认情况下，此事件处于禁用状态，应用程序可以通过调用 API esp_wifi_set_event_mask() 启用。启用后，每当 AP 接收到 probe request 时都将引发此事件。
**Chapter 4. API 指南**

**WIFI_EVENT_STA_BEACON_TIMEOUT**

如果 station 在 inactive 时间内未收到所连接 AP 的 beacon，将发生 beacon 超时，将引发此事件。inactive 时间通过调用函数 `esp_wifi_set_inactive_time()` 设置。

**WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START**

非连接模块在 Interval 开始时触发此事件。请参考非连接模块功耗管理。

**4.3.3 ESP32 Wi-Fi station 一般情况**

下图为 station 模式下的宏观场景，其中包含不同阶段的具体描述：

1. Wi-Fi/LwIP 初始化阶段

   • s1.1：主任务通过调用函数 `esp_netif_init()` 创建一个 LwIP 核心任务，并初始化 LwIP 相关工作。
   • s1.2：主任务通过调用函数 `esp_event_loop_create()` 创建一个系统事件任务，并初始化应用程序事件的回调函数。在此情况下，该回调函数唯一的作用就是将事件中继到应用程序任务中。
   • s1.3：主任务通过调用函数 `esp_netif_create_default_wifi_ap()` 或 `esp_netif_create_default_wifi_sta()` 创建有 TCP/IP 堆栈的默认网络接口实例绑定 station 或 AP。
   • s1.4：主任务通过调用函数 `esp_wifi_init()` 创建 Wi-Fi 驱动程序任务，并初始化 Wi-Fi 驱动程序。
   • s1.5：主任务通过调用 OS API 创建应用程序任务。

推荐按照 s1.1 ~ s1.5 的步骤顺序基于 Wi-Fi/LwIP 的应用程序进行初始化。但这一顺序 并非 强制，您可以在第 s1.1 步创建应用程序任务，然后在该应用程序任务中进行所有其它初始化操作。不过，如果您应用程序任务依赖套接字，那么在初始化阶段创建应用程序任务可能并不适用。此时，您可以在接收到 IP 后再进行任务创建。

2. Wi-Fi 配置阶段

Wi-Fi 驱动程序初始化成功后，可以进入到配置阶段。该场景下，Wi-Fi 驱动程序处于 station 模式。因此，首先您需要调用函数 `esp_wifi_set_mode()` (WIFI_MODE_STA) 将 Wi-Fi 模式配置为 station 模式。可通过调用其它 `esp_wifi_set_xxx` API 进行更多设置，例如：协议模式、国家代码、带宽等。请参阅 ESP32 Wi-Fi 配置。

一般情况下，我们会在建立 Wi-Fi 连接之前配置 Wi-Fi 驱动程序，但这 并非 强制要求，也就是说，只要 Wi-Fi 驱动程序已成功初始化，您可以在任意阶段进行配置。但是，如果您的 Wi-Fi 在建立连接后不需要更改配置，则应先在该阶段完成配置。因为调用配置 API (例如 `esp_wifi_set_protocol()` ) 将会导致 Wi-Fi 连接断开，为您的操作带来不便。

如果 menuconfig 已使能 Wi-Fi NVS flash，刚不论当前阶段还是后续的 Wi-Fi 配置信息都将被存储至该 flash 中。那么，当主板上电/重新启动时，就不需从头开始配置 Wi-Fi 驱动程序。您只需调用函数 `esp_wifi_get_xxx` API 获取之前存储的配置信息。当然，如果不想使用之前的配置，您依然可以重新配置 Wi-Fi 驱动程序。

3. Wi-Fi 启动阶段

   • s3.1：调用函数 `esp_wifi_start()` 启动 Wi-Fi 驱动程序。
   • s3.2：Wi-Fi 驱动程序将事件 `WIFI_EVENT_STA_START` 发布到事件任务中，然后，事件任务将执行一些正常操作并调用应用程序的事件回调函数。
   • s3.3：应用程序的事件回调函数将事件 `WIFI_EVENT_STA_START` 发布到应用程序任务中。您此时调用函数 `esp_wifi_connect()` 进行 Wi-Fi 连接。当然，您也可以等待在 `WIFI_EVENT_STA_START` 事件发生后的其它阶段再调用此函数。
图 5.5: station 模式下 Wi-Fi 事件场景示例
4. Wi-Fi 连接阶段

- s4.1: 调用函数 `esp_wifi_connect()` 后，Wi-Fi 驱动程序将启动内部扫描/连接过程。
- s4.2: 如果内部扫描/连接过程成功，将产生 `WIFI_EVENT_STA_CONNECTED` 事件。然后，事件任务将启动 DHCP 客户端服务，最终触发 DHCP 程序。
- s4.3: 在此情况下，应用程序的事件回调函数将收到 `WIFI_EVENT_STA_CONNECTED` 事件中继到应用程序任务中。通常，应用程序不需进行操作，而您可以执行任何动作，例如：打印日志等。

步骤 s4.2 中 Wi-Fi 连接可能会由于某些原因而失败，例如：密码错误、未找到 AP 等。这种情况下，将引发 `WIFI_EVENT_STA_DISCONNECTED` 事件并提示连接错误原因。有关如何处理中断 Wi-Fi 连接的事件，请参阅第6章 6 的描述。

5. Wi-Fi 获取 IP 阶段

- s5.1: 一旦步骤 4.2 中的 DHCP 客户端初始化完成，Wi-Fi 驱动程序将进入获取 IP 阶段。
- s5.2: 如果 Wi-Fi 成功从 DHCP 服务器接收到 IP 地址，则将引发 `IP_EVENT_STA_GOT_IP` 事件，事件任务将执行正常处理。
- s5.3: 应用程序的事件回调函数将事件 `IP_EVENT_STA_GOT_IP` 中继到应用程序任务中。对于那些基于 LwIP 构建的应用程序，此事件较为特殊，因为它意味着应用程序已准备好，可以开始任务，例如：创建 TCP/UDP 套接字。此时较为容易犯的一个错误就是在接收到 `IP_EVENT_STA_GOT_IP` 事件之前就初始化套接字。切记在接收到 IP 之前启动任何套接字相关操作。

6. Wi-Fi 断开阶段

- s6.1: 当 Wi-Fi 因为某些原因（例如：AP 掉电、RSSI 较弱等）连接中断时，将产生 `WIFI_EVENT_STA_DISCONNECTED` 事件。此事件也可能在上文步骤 3 中发生。在这里，事件任务将通知 LwIP 任务清除/移除所有 UDP/TCP 连接。然后，所有应用程序套接字都将处于错误状态。也就是说，`WIFI_EVENT_STA_DISCONNECTED` 事件发生时，所有套接字都无法正常工作。
- s6.2: 上述情况下，应用程序的事件回调函数将将 `WIFI_EVENT_STA_DISCONNECTED` 事件中继到应用程序任务中。推荐您调用函数 `esp_wifi_connect()` 重新连接 Wi-Fi，关闭所有套接字，并在必要时重新创建套接字。请参阅 `WIFI_EVENT_STA_DISCONNECTED`。

7. Wi-Fi IP 更改阶段

- s7.1: 如果 IP 地址发生更改，将引发 `IP_EVENT_STA_GOT_IP` 事件，其中 “ip_change” 被置为 “true”。
- s7.2: 此事件对应用程序至关重要。这一事件发生时，适合关闭所有已创建的套接字并进行重新创建。

8. Wi-Fi 清理阶段

- s8.1: 调用函数 `esp_wifi_disconnect()` 断开 Wi-Fi 连接。
- s8.2: 调用函数 `esp_wifi_stop()` 终止 Wi-Fi 驱动程序。
- s8.3: 调用函数 `esp_wifi_deinit()` 清理 Wi-Fi 驱动程序。

4.33.8 ESP32 Wi-Fi AP 一般情况

下图为 AP 模式下的宏观场景，其中包含不同阶段的具体描述：

4.33.9 ESP32 Wi-Fi 扫描

目前，仅 station 或 station/AP 共存模式支持 `esp_wifi_scan_start()` API。
图 56: AP 模式下 Wi-Fi 事件场景示例
### 扫描类型

<table>
<thead>
<tr>
<th>模式</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>主动扫描</td>
<td>通过发送 probe request 进行扫描。该模式为默认的扫描模式。</td>
</tr>
<tr>
<td>被动扫描</td>
<td>不发送 probe request。跳至某一特定信道并等待 beacon。应用程序可通过 wifi_scan_config_t 中的 scan_type 字段使能被动扫描。</td>
</tr>
<tr>
<td>前端扫描</td>
<td>在 station 模式下 Wi-Fi 未连接时，可进行前端扫描。Wi-Fi 驱动程序决定进行前端扫描还是后端扫描，应用程序无法配置这两种模式。</td>
</tr>
<tr>
<td>后端扫描</td>
<td>在 station 模式或 station/AP 共存模式下 Wi-Fi 已连接时，可进行后端扫描。Wi-Fi 驱动程序决定进行前端扫描还是后端扫描，应用程序无法配置这两种模式。</td>
</tr>
<tr>
<td>全信道扫描</td>
<td>扫描所有信道。wifi_scan_config_t 中的 channel 字段为 0 时，当前模式为全信道扫描。</td>
</tr>
<tr>
<td>特定信道扫描</td>
<td>只扫描特定的信道。wifi_scan_config_t 中的 channel 字段为 1-14 时，当前模式为特定信道扫描。</td>
</tr>
</tbody>
</table>

上表中的扫描模式可以任意组合，因此共有 8 种不同扫描方式:

- 全信道后端主动扫描
- 全信道后端被动扫描
- 全信道前端主动扫描
- 全信道前端被动扫描
- 特定信道后端主动扫描
- 特定信道后端被动扫描
- 特定信道前端主动扫描
- 特定信道前端被动扫描

### 扫描配置

扫描类型与其他扫描属性通过函数 esp_wifi_scan_start() 进行配置。下表详细描述了函数 wifi_scan_config_t 各字段信息。

<table>
<thead>
<tr>
<th>字段</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssid</td>
<td>如果该字段的值不为 NULL，则仅可扫描到具有相同 SSID 值的 AP。</td>
</tr>
<tr>
<td>bssid</td>
<td>如果该字段的值不为 NULL，则仅可扫描到具有相同 BSSID 值的 AP。</td>
</tr>
<tr>
<td>channel</td>
<td>如果该字段值为 0，将进行全信道扫描；反之，将针对特定信道进行扫描。</td>
</tr>
<tr>
<td>show_hidden</td>
<td>如果该字段值为 0，本次扫描将忽略具有隐藏 SSID 的 AP；反之，这些 AP 也会在扫描时被视为正常 AP。</td>
</tr>
<tr>
<td>scan_type</td>
<td>如果该字段值为 WIFI_SCAN_TYPE_ACTIVE，则本次扫描为自动扫描；反之，将被视为被动扫描。</td>
</tr>
<tr>
<td>scan_time</td>
<td>该字段用于控制每个信道的扫描时间。被扫描时，scan_time.active 字段负责为每个信道指定扫描时间。主扫描时，每个信道的扫描时间如下所示。其中，min 代表 scan_time.active_min，max 代表 scan_time.active_max。</td>
</tr>
</tbody>
</table>

- min=0, max=0：每个信道的扫描时间为 120 ms。
- min>0, max=0：每个信道的扫描时间为 120 ms。
- min=0, max>0：每个信道的扫描时间为 max ms。
- min>0, max>0：每个信道扫描的最短时间为 min ms。如果在这段时间内未找到 AP，将跳转至下一个信道。如这时间内找到 AP，则该信道的扫描时间为 max ms。

如希望提升 Wi-Fi 扫描性能，则可修改上述两个参数。

调用 API esp_wifi_set_config() 可全局配置一些扫描属性，请参阅 station 基本配置。
在所有信道中扫描全部 AP（前端）

场景:

![图57: 所有Wi-Fi信道的前端扫描](image)

上述场景中描述了全信道前端扫描过程。仅 station 模式支持前端扫描，该模式下 station 未连接任何 AP。前端扫描还是后端扫描完全由 Wi-Fi 驱动程序决定，应用程序无法配置这一模式。

详细描述:

扫描配置阶段

- s1.1: 如果默认的国家信息有误，调用函数 `esp_wifi_set_country()` 进行配置。请参阅 Wi-Fi 国家/地区代码。
- s1.2: 调用函数 `esp_wifi_scan_start()` 配置扫描信息，可参阅扫描配置。该场景为全信道扫描，将 SSID/BSSID/channel 设置为 0 即可。

Wi-Fi 驱动程序内部扫描阶段
扫描完成后事件处理阶段

s.3.1: 当所有信道扫描全部完成后，将产生 WIFI_EVENT_SCAN_DONE 事件。
s.3.2: 应用程序的事件回调函数告知应用程序任务已接收 WIFI_EVENT_SCAN_DONE 事件。调用
函数 esp_wifi_scan_get_ap_num() 获取在本次扫描中找到的 AP 数量，然后，分配出足够
的事物槽，并调用函数 esp_wifi_scan_get_ap_records() 获取 AP 记录。请注意，一旦调
用 esp_wifi_scan_get_ap_records()，Wi-Fi 驱动程序中的 AP 记录将被释放。但是，请不要
在单个扫描完成事件中重复调用 esp_wifi_scan_get_ap_records()。反之，如果扫描完
成事件发生后未调用 esp_wifi_scan_get_ap_records()，则 Wi-Fi 驱动程序中的 AP 记录不
会被释放。因此，请务必确保调用函数 esp_wifi_scan_get_ap_records()，且仅调用一次。

在所有信道上扫描全部 AP（后端）

场景：
上述场景为一次信道后端扫描。与在所有信道中扫描全部 AP（前端）相比，信道后端扫描的不同
之处在于：在跳至下一个信道之前，Wi-Fi 驱动程序会先返回主信道停留 30 ms，以便 Wi-Fi 连接有一定的
时间发送/接收数据。

在所有信道中扫描特定 AP

场景：
该扫描过程与在所有信道中扫描全部 AP（前端）相似。区别在于：
- s.1.1: 在步骤 1.2 中，目标 AP 将配置为 SSID/BSSID。
- s.2.1 ~ s.2.N: 每当 Wi-Fi 驱动程序扫描某个 AP 时，它将检查该 AP 是否为目标 AP。如果本次扫描
类型为 WIFI_FAST_SCAN，且已确认没有找到目标 AP，则将产生扫描完成事件，同时结束本次扫描；
反之，扫描将继续。请注意，第一个扫描的信道可能不是信道 1，因为 Wi-Fi 驱动程序会优化扫描
顺序。

如果有多个匹配目标 AP 信息的 AP，例如：碰到扫描到两个 SSID 为 “ap” 的 AP。如果本
次扫描类型为 WIFI_FAST_SCAN，则仅可扫描到第一个扫描到的“ap”；如果本次扫描类型为
WIFI_ALL_CHANNEL_SCAN，则两个“ap”都将被找到，且 station 将根据配置规则连接至其需要连接
的“ap”，请参阅 station 基本配置。

您可以在任意信道中扫描某个特定的 AP，或扫描该信道中的所有 AP，这两种扫描过程也较为相似。

在 Wi-Fi 连接模式下扫描

调用函数 esp_wifi_connect() 后，Wi-Fi 驱动程序将首先尝试扫描已配置的 AP，Wi-Fi 连接模式下的
扫描过程与在所有信道中扫描特定 AP 过程相同，但连接模式下扫描结束后将不会产生扫描完成事件。如
果已找到目标 AP，则 Wi-Fi 驱动程序将开始 Wi-Fi 连接；反之，将产生 WIFI_EVENT_STA_DISCONNECTED
事件。请参阅在所有信道中扫描特定 AP。

在禁用模式下扫描

如果函数 esp_wifi_scan_start() 中的禁用参数为 “true”，则本次扫描为禁用模式下的扫描。在该
次扫描完成之前，应用程序任务都将被禁用。禁用模式下的扫描和正常扫描相似，不同之处在于，禁用
模式下扫描完成之后将不会出现扫描完成事件。
图 58: 所有 Wi-Fi 信道的后端扫描
图 59: 扫描特定的 Wi-Fi 信道
并行扫描

有时，可能会有两个应用程序任务同时调用函数 `esp_wifi_scan_start()`，或者某个应用程序任务在获取扫描完成事件之前再次调用了函数 `esp_wifi_scan_start()`。这两种情况都可能会发生。但是，Wi-Fi 驱动程序并不足以支持多个并行的扫描。因此，应避免上述并行扫描。随着 ESP32 的 Wi-Fi 功能不断提升，未来的版本中可能会增加并行扫描支持。

连接 Wi-Fi 时扫描

如果 Wi-Fi 正在连接，则调用函数 `esp_wifi_scan_start()` 后扫描将立即失败，因为 Wi-Fi 连接优先级高于扫描。如果扫描是因为 Wi-Fi 连接而失败的，此时推荐采取的策略为：等待一段时间后重试。因为一旦 Wi-Fi 连接完成后，扫描将立即成功。

但是，延时重试策略并非万无一失。试想以下场景：

- 如果 `station` 正在连接一个不存在的 AP，或正在使用错误的密码，错误一个 AP，此时将产生事件 `WIFI_EVENT_STA_DISCONNECTED`。
- 接收到断开连接事件后，应用程序调用函数 `esp_wifi_connect()` 进行重新连接。
- 而另一个应用程序任务（如控制任务）调用了函数 `esp_wifi_scan_start()` 进行扫描。这种情况下，每次扫描都会立即失败，因为 `station` 一直处于正在连接状态。
- 扫描失败后，应用程序将等待一段时间后进行重新扫描。

上述场景中的扫描永远不会成功，因为 Wi-Fi 一直处于连接过程中。因此，如果您的应用程序也可以发生相似的场景，那么就需要为其配置一个更佳的重新连接策略。例如；

- 应用程序可以定义一个连续重新连接次数的最大值，当重新连接的次数达到这个最大值时，立刻停止重新连接。
- 应用程序可以在首轮连续重新连接 N 次后立即进行重新连接，然后延迟一段时间后进行下一次重新连接。

可以给应用程序定义其特殊的重新连接策略，以防止扫描无法成功。请参阅 Wi-Fi 重新连接。

4.33.10 ESP32 Wi-Fi station 连接场景

该场景仅针对在扫描阶段只找到一个目标 AP 的情况，对于多个相同 SSID AP 的情况，请参阅找到多个 AP 时的 ESP32 Wi-Fi station 连接。

通常，应用程序无需关心这一连接过程。如感兴趣，可参看下述简介。

场景：

扫描阶段

- s1.1: Wi-Fi 驱动程序开始在 “Wi-Fi 连接” 模式下扫描。详细信息请参阅在 Wi-Fi 连接模式下扫描。
- s1.2: 如果未找到目标 AP，将产生 `WIFI_EVENT_STA_DISCONNECTED` 事件，且原因代码为 `WIFI_REASON_NO_AP_FOUND`。请参阅 Wi-Fi 原因代码。

认证阶段

- s2.1: 发送认证请求数据包并使能认证计时器。
- s2.2: 如果在认证计时器超时之前未接收到认证响应数据包，将产生 `WIFI_EVENT_STA_DISCONNECTED` 事件，且原因代码为 `WIFI_REASON_AUTH_EXPIRE`。请参阅 Wi-Fi 原因代码。
- s2.3: 接收到认证响应数据包，且认证计时器终止。
- s2.4: AP 在响应中拒绝认证且产生 `WIFI_EVENT_STA_DISCONNECTED` 事件，原因代码为 `WIFI_REASON_AUTH_FAIL` 或为 AP 指定的其它原因。请参阅 Wi-Fi 原因代码。
Chapter 4. API 指南

图 60. Wi-Fi Station 连接过程
关联阶段

- s3.1: 发送关联请求并使能关联计时器。
- s3.2: 如果在关联计时器超时之前未接收到关联响应，将产生 WIFI_EVENT_STA_DISCONNECTED 事件，且原因代码为 WIFI_REASON_ASSOC_EXPIRE。请参阅 Wi-Fi 原因代码。
- s3.3: 接收到关联响应，且关联计时器终止。
- s3.4: AP 在响应中拒绝关联且产生 WIFI_EVENT_STA_DISCONNECTED 事件，原因代码将在关联响应中指定。请参阅 Wi-Fi 原因代码。

四次握手阶段

- s4.1: 使能握手定时器，定时器终止之前未接收到 1/4 EAPOL，此时将产生 WIFI_EVENT_STA_DISCONNECTED 事件，且原因代码为 WIFI_REASON_HANDSHAKE_TIMEOUT。请参阅 Wi-Fi 原因代码。
- s4.2: 接收到 1/4 EAPOL。
- s4.3: station 回复 2/4 EAPOL。
- s4.4: 如果在握手定时器终止之前未接收到 3/4 EAPOL，将产生 WIFI_EVENT_STA_DISCONNECTED 事件，且原因代码为 WIFI_REASON_HANDSHAKE_TIMEOUT，请参阅 Wi-Fi 原因代码。
- s4.5: 接收到 3/4 EAPOL。
- s4.6: station 回复 4/4 EAPOL。
- s4.7: station 产生 WIFI_EVENT_STA_CONNECTED 事件。

Wi-Fi 原因代码

下表罗列了 ESP32 中定义的原因代码，其中，第一列为 esp_wifi_types.h 中定义的宏名称，名称中省去了前缀 WIFI_REASON，也就是说，名称 UNSPECIFIED 实际应为 WIFI_REASON_UNSPECIFIED，以此类推。第二列为原因代码的相应数值。第三列为该原因映射到 IEEE 802.11-2020 中 9.4.1.7 段的标准值。（更多详细信息，请参阅前文描述。）最后一列为此原因的描述。

<table>
<thead>
<tr>
<th>原因代码</th>
<th>数值</th>
<th>映射值</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNSPECIFIED</td>
<td>1</td>
<td>1</td>
<td>出现内部错误，例如：内存已满，内部发送失败，或该原因已被远端接收等。</td>
</tr>
</tbody>
</table>
| AUTH_EXPIRE | 2 | | 先前的 authentication 已失效。对于 ESP station，出现以下情况时将报告该代码：
- authentication 超时；
- 从 AP 接收到该代码。
对于 ESP AP，出现以下情况时将报告该代码：
- 在过去五分钟之内，AP 未从 station 接收到任何数据包；
- 由于调用了函数 esp_wifi_stop() 导致 AP 终止；
- 由于调用了函数 esp_wifi_deauth_sta() 导致 station 的 authentication 取消。 |
| AUTH_LEAVE | 3 | | authentication 取消，因为发送 station 正在离开（或已经离开）。
对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。 |

下页继续
<table>
<thead>
<tr>
<th>原因代码</th>
<th>数值</th>
<th>映射值</th>
<th>描述</th>
</tr>
</thead>
</table>
| AS-SOC_EXPIRE | 4 | 4 | 因为 AP 无法同时处理所有当前已关联的 STA。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。
- 对于 ESP AP，出现以下情况时将报告该代码：
  - 经过五分钟，AP 未从 station 接收到任何数据包；
  - 由于调用了函数 esp_wifi_stop() 导致 AP 终止；
- 由于调用了函数 esp_wifi_deauth_sta() 导致 station 的 authentication 取消。 |
| AS-SOC_TOOMANY | 5 | 5 | association 取消，因为 AP 无法同时处理所有当前已关联的 STA。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。
- 对于 ESP AP，出现以下情况时将报告该代码：
  - 与 AP 相关联的 station 数量已到达 AP 可支持的最大值。 |
| NOT_AUTHED | 6 | 6 | 从一个未认证 station 接收到 class-2 frame。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。
- 对于 ESP AP，出现以下情况时将报告该代码：
  - AP 从一个未认证 station 接收到数据包。 |
| NOT_ASSOCED | 7 | 7 | 从一个未关联 station 接收到的 class-3 frame。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。
- 对于 ESP AP，出现以下情况时将报告该代码：
  - AP 从未关联 station 接收到数据包。 |
| AS-SOC_LEAVE | 8 | 8 | association 取消，因为发送 station 正在离开（或已经离开）BSS。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。
- 由于调用 esp_wifi_disconnect() 和其它 API，station 断开连接。 |
| AS-SOC_NOT_AUTHED | 9 | 9 | station 的 reassociation 请求未被响应 station 认证。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。
- 对于 ESP AP，出现以下情况时将报告该代码：
  - AP 从一个已关联、但未认证的 station 接收到数据包。 |
| DIS-AS-SOC_PWRCAP_BAD | 10 | 10 | association 取消，因为无法接收功率能力（Power Capability）元素中的信息。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。 |
| DIS-AS-SOC_SUPCHAN_BAD | 11 | 11 | association 取消，因为无法接收支持的信道（Supported Channels）元素中的信息。对于 ESP station，出现以下情况时报告该代码：
- 从 AP 接收到该代码。 |
<table>
<thead>
<tr>
<th>原因代码</th>
<th>数值</th>
<th>映射值</th>
<th>描述</th>
</tr>
</thead>
</table>
| IE_INVALID | 13 | | 无效元素或内容不符合Wi-Fi协议帧格式(Frame formats)章节所描述的标准的元素。对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。
对于ESP AP，出现以下情况时将报告该代码：
- AP解析了一个错误的WPA或RSN IE。 |
| MIC_FAILURE | 14 | | 消息完整性代码(MIC)出错。对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。 |
| 4WAY_HANDSHAKE_TIMEOUT | | | 四次握手超时。由于某些历史原因，在ESP中该原因代码实为WIFI_REASON_HANDSHAKE_TIMEOUT。
对于ESP station，出现以下情况时报告该代码：
- 握手超时。
- 从AP接收到该代码。 |
| GROUP_CIPHER_INVALID | | | 组密钥(Group-Key)握手超时。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。 |
| IE_IN_4WAY_DIFFERS17 | | | 四次握手中产生的元素与(re-)association后的request/probe以及response/beacon frame中的信息不同。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。
- station发现四次握手的IE与(re-)association后的request/probe以及response/beacon frame中的IE不同。 |
| GROUP_CIPHER_INVALID | | | 无效组密钥。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。 |
| PAIRWISE_CIPHER_INVALID | 19 | 19 | 无效或对密钥。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。 |
| AKMP_INVALID | 20 | | 无效AKMP。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。 |
| UN-SUPP_RSN_IE_VERSION | 21 | 21 | 不支持的RSNE版本。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。 |
| IN-VALID_RSN_IE_CAP | 22 | 22 | 无效的RSNE性能。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。 |
| 802_1X_2AUTH_FAILURE | 23 | | IEEE 802.1X authentication失败。
对于ESP station，出现以下情况时报告该代码：
- 从AP接收到该代码。
对于ESP AP，出现以下情况时将报告该代码：
- IEEE 802.1X authentication失败。 |
<table>
<thead>
<tr>
<th>原因代码</th>
<th>数值</th>
<th>映射值</th>
<th>描述</th>
</tr>
</thead>
</table>
| CI-PHER_SUITE_REJECTED | 24     | 24     | 因安全策略，安全密钥算法套件（cipher suite）被拒。
|           |        |        | 用于设备使用，若该情况时报告该代码：
|           |        |        | 从 AP 接收到该代码。                       |
| TDLS_PEER_UNREACHABLE | 31     | 31     | 通过 TDLS 直连无法到达 TDLS 对端 STA，导致 TDLS 直连中断。
| TDLS_UNSPECIFIED | 32     | 32     | 未明原因的 TDLS 直连中断。                |
| SSP_REQUESTED_DISASSOC | 33     | 33     | association 取消，由于会话被 SSP request 终止。 |
| NO_SSP_PROAMING_AGREEMENT | 34     | 34     | association 取消，由于缺乏 SSP 漫游认证。 |
| BAD_CIPHER_OR_AKM | 35     | 35     | 请求的服务被拒绝，由于 SSP 密钥套件或者 AKM 的需求。 |
| NOT_AUTHORIZED_TMISS_LO | 36     | 36     | 服务请求未得到授权。                    |
| SERVICE_CHANGE_PRECLUDES_TS | 37     | 37     | QoS 被删除，原因是：BSS 服务特性或者运行模式改变导致 QoS AP 缺少足够的带宽给 QoS STA 使用（例如：一个 HT BSS 从 40 MHz 的信道切换到 20 MHz 的信道）。 |
| UNSPECIFIED_QOS | 38     | 38     | association 取消，由于不明确的 QoS 相关原因。 |
| NOT_ENOUGH_BANDWIDTH | 39     | 39     | association 取消，由于 QoS AP 缺少足够的带宽给该 QoS STA 使用。 |
| MISSING_ACKS | 40     | 40     | association 取消，原因：有大量的帧需要被确认，但由于 AP 传输或者糟糕的信道条件而没有被确认。 |
| EXCEEDED_TXOP | 41     | 41     | association 取消，由于 STA 的传输超过了 TXOPs 的限制。 |
| STA_LEAVEING | 42     | 42     | 请求 STA 离开了 BSS 或者重置了。 |
| END_STA | 43     | 43     | 请求 STA 不再使用该流或者会话。 |
| UNKNOWN | 44     | 44     | 请求 STA 使用一种尚未完成的机制接收帧。 |
| TIME-OUT | 45     | 45     | 对端 STA 的请求超时。                    |
| RESERVED | 46     | 46     | 在 Disassociation 表中：已达到授权访问限制。 |
| PEER_INITIATED | 47     | 47     | 在 Disassociation 表中：外部服务需求。 |
| IN-VALID_FT_ACTION_FRAME_COUNT | 48     | 48     | 无效的 FT Action 帧计数。 |
| IN-VALID_PMKID | 49     | 49     | 无效的成对主密钥标识符（PMKID）。 |
| IN-VALID_MDE | 50     | 50     | 无效的 MDE。 |
| IN-VALID_FTE | 51     | 51     | 无效的 FTE。 |
| TRANSMISSION_LINK_ESTABLISHMENT_FAILED | 52     | 52     | 在备用信道中建立传输链路失败。 |
| ALTERATIVE_CHANNEL_OCCUPIED | 53     | 53     | 备用信道被占用。 |
| BEACON_TIMEOUT | 54     | 54     | 保留。 |
| NO_AP | 55     | 55     | 保留。 |

下页继续
表 29 - 续上页

<table>
<thead>
<tr>
<th>原因代码</th>
<th>数值</th>
<th>映射值</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_FAIL</td>
<td>201</td>
<td>保留</td>
<td>乐鑫特有的 Wi-Fi 原因代码：authentication 失败，但并非由超时而引发。</td>
</tr>
<tr>
<td>AS-SOC_FAIL</td>
<td>203</td>
<td>保留</td>
<td>乐鑫特有的 Wi-Fi 原因代码：association 失败，但并非由 ASSOC_EXPIRE 或 ASSOC_TOOMANY 引发。</td>
</tr>
<tr>
<td>HANDSHAKE_TIMEOUT</td>
<td>204</td>
<td>保留</td>
<td>乐鑫特有的 Wi-Fi 原因代码：握手失败，与 WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT 中失败原因相同。</td>
</tr>
<tr>
<td>CONNECTION_FAIL</td>
<td>205</td>
<td>保留</td>
<td>乐鑫特有的 Wi-Fi 原因代码：AP 连接失败。</td>
</tr>
</tbody>
</table>

与密码错误有关的 Wi-Fi 原因代码

下表列出了与密码错误相关的 Wi-Fi 原因代码。

<table>
<thead>
<tr>
<th>原因代码</th>
<th>数值</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>4WAY_HANDSHAKE_TIMEOUT</td>
<td>密码错误出现这个原因代码的场景有如下两个：</td>
<td></td>
</tr>
<tr>
<td>NO_AP_FOUNDED</td>
<td>STA 在连接加密的 AP 的时候没有输入密码</td>
<td></td>
</tr>
<tr>
<td>HANDSHAKE_TIMEOUT</td>
<td>STA 在连接非加密的 AP 的时候输入了密码</td>
<td></td>
</tr>
<tr>
<td>HANDSHAKE_TIMEOUT</td>
<td>STA 在连接非加密的 AP 的时候输入了密码</td>
<td></td>
</tr>
</tbody>
</table>

与低 RSSI 有关的 Wi-Fi 原因代码

下表列出了与低 RSSI 相关的 Wi-Fi 原因代码。

<table>
<thead>
<tr>
<th>原因代码</th>
<th>数值</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_AP_FOUNDED</td>
<td>低 RSSI 导致 station 无法扫描到目标 AP</td>
<td></td>
</tr>
<tr>
<td>HANDSHAKE_TIMEOUT</td>
<td>站点超时。</td>
<td></td>
</tr>
</tbody>
</table>

4.33.11 找到多个 AP 时的 ESP32 Wi-Fi station 连接

该场景与 ESP32 Wi-Fi station 连接场景相似，不同之处在于该场景中不会产生 WIFI_EVENT_STA_DISCONNECTED 事件，除非 station 无法连接到所有找到的 AP。

4.33.12 Wi-Fi 重新连接

出于多种原因，station 可能会断开连接，例如：连接的 AP 重新启动等。应用程序应负责重新连接。推荐使用的方法为：在接收到 WIFI_EVENT_STA_DISCONNECTED 事件后调用函数 esp_wifi_disconnect()。但有时，应用程序需要更复杂的方式进行重新连接：

- 如果断开连接事件是由调用函数 esp_wifi_disconnect() 引发的，那么应用程序可能不需要进行重新连接。
- 如果 station 随时可能调用函数 esp_wifi_scan_start() 开始扫描，此时就需要一个更佳的重新连接方法，请参阅连接 Wi-Fi 时扫描。
另一点需要注意的是，如果存在多个具有相同 SSID 的 AP，那么重新连接后可能不会连接到之前的同一个 AP。重新连接时，station 将永远选择最佳的 AP 进行连接。

4.33.13 Wi-Fi beacon 超时

ESP32 使用 beacon 超时机制检测 AP 是否活跃。如果 station 在 inactive 时间内未收到所连接 AP 的 beacon，将发生 beacon 超时。 inactive 通过调用函数 `esp_wifi_set_inactive_time()` 设置。

beacon 超时发生后，station 将向 AP 发送 5 个 probe request。如果仍未从 AP 接收到 probe response 或 beacon，station 将与 AP 断开连接并产生 `WIFI_EVENT_STA_DISCONNECTED` 事件。

需要注意的是，扫描过程中会重置 beacon 超时使用的定时器，即扫描过程会影响 `WIFI_EVENT_STA_BEACON_TIMEOUT` 事件的触发。

4.33.14 ESP32 Wi-Fi 配置

使能 Wi-Fi NVS 时，所有配置都将存储到 flash 中；反之，请参阅 Wi-Fi NVS Flash。

Wi-Fi 模式

调用函数 `esp_wifi_set_mode()` 设置 Wi-Fi 模式。

<table>
<thead>
<tr>
<th>模式</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIFI_MODE_NULL</td>
<td>NULL 模式：此模式下，内部数据结构不分配给 station 和 AP，同时，station 和 AP 接口不会为发送/接收 Wi-Fi 数据进行初始化。通常，此模式用于 Sniffer，或者您不想通过调用函数 <code>esp_wifi_deinit()</code> 卸载整个 Wi-Fi 驱动程序来同时停止 station 和 AP。</td>
</tr>
<tr>
<td>WIFI_MODE_STA</td>
<td>station 模式：此模式下，<code>esp_wifi_start()</code> 将初始化内部 station 数据，同时 station 接口准备发送/接收 Wi-Fi 数据。调用函数 <code>esp_wifi_connect()</code> 后，station 将连接到目标 AP。</td>
</tr>
<tr>
<td>WIFI_MODE_AP</td>
<td>AP 模式：在此模式下，<code>esp_wifi_start()</code> 将初始化内部 AP 数据，同时 AP 接口准备发送/接收 Wi-Fi 数据。随后，Wi-Fi 驱动程序开始广播 beacon，AP 即可与其它 station 连接。</td>
</tr>
<tr>
<td>WIFI_MODE_APSTA</td>
<td>station/AP 共存模式：在此模式下，<code>esp_wifi_start()</code> 将同时初始化 station 和 AP。该步骤在 station 模式和 AP 模式下完成。请注意 ESP station 所连外部 AP 的信道优先于 ESP AP 信道。</td>
</tr>
</tbody>
</table>

station 基本配置

API `esp_wifi_set_config()` 可用于配置 station。配置的参数信息会保存到 NVS 中。下表详细介绍各个字段。
### Chapter 4. API 指南

<table>
<thead>
<tr>
<th>字段</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssid</td>
<td>station 想要连接的目标 AP 的 SSID。</td>
</tr>
<tr>
<td>password</td>
<td>目标 AP 的密码。</td>
</tr>
<tr>
<td>scan_method</td>
<td>WIFI_FAST_SCAN 模式下，扫描到一个匹配的 AP 时即结束。WIFI_ALL_CHANNEL_SCAN 模式下，在所有信道扫描所有匹配的 AP。默认扫描模式是 WIFI_FAST_SCAN。</td>
</tr>
<tr>
<td>bssid_set</td>
<td>如果 bssid_set 为 0，station 连接 SSID 与 “ssid” 字段相同的 AP，同时忽略字段 “bssid”。其他情况下，station 连接 SSID 与 “ssid” 字段相同，BSSID 与 “bssid” 字段也相同的 AP。</td>
</tr>
<tr>
<td>bssid</td>
<td>只有当 bssid_set 为 1 时有效。见字段 “bssid_set”。</td>
</tr>
<tr>
<td>channel</td>
<td>该字段为 0 时，station 扫描信道 1 ~ N 寻找目标 AP；否则，station 首先扫描 SSID 与 channel 字段相同的信道，再扫描其他信道。如果您不知道目标 AP 在哪个信道，请将该字段设置为 0。</td>
</tr>
<tr>
<td>sort_method</td>
<td>该字段仅用于 WIFI_ALL_CHANNEL_SCAN 模式。如果设置为 WIFI_CONNECT_AP_BY_SIGNAL，所有匹配的 AP 将会按照信号强度排序，信号最好的 AP 会被首选。如果 station 想要连接 ssid 为 “apxx” 的 AP，并扫描到两个这样的 AP。第一个 AP 的信号为 -90 dBm，第二个 AP 的信号为 -100 dBm，station 首先选择连接第二个 AP。除非失败，才会选择第一个。如果设置为 WIFI_CONNECT_AP_BY_SECURITY，所有匹配的 AP 会按照安全性排序。比如，如果 station 想要连接 ssid 为 “apxx” 的 AP，并扫描到两个这样的 AP。第一个 AP 为开放式，第二个 AP 为 WPA2 加密，station 首先选择连接第二个 AP。除非失败，才会选择第一个。</td>
</tr>
<tr>
<td>threshold</td>
<td>该字段用来筛选找到的 AP，如果 AP 的 RSSI 或安全模式小于配置的阈值，则不会被连接。如果 RSSI 设置为 0，则表示默认阈值，默认 RSSI 阈值为 -127 dBm。如果 authmode 阈值设置为 0，则表示默认阈值，默认 authmode 阈值无授权。</td>
</tr>
</tbody>
</table>

**注意：** WEP/WPA 安全模式在 IEEE802.11-2016 协议中已弃用，建议不要使用。可使用 authmode 阈值代替，通过将 threshold.authmode 设置为 WIFI_AUTH_WPA2_PSK 使用 WPA2 模式

### AP 基本配置

API esp_wifi_set_config() 可用于配置 AP。配置的参数信息会保存在 NVS 中。下表详细介绍了各个字段。

<table>
<thead>
<tr>
<th>字段</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssid</td>
<td>指 AP 的 SSID。如果 ssid[0] 和 ssid[1] 均为 0xFF，AP 默认 SSID 为 ESP_aabbcc，”aabbcc” 是 AP MAC 的最后三个字节。</td>
</tr>
<tr>
<td>password</td>
<td>AP 的密码。如果身份验证模式为 WIFI_AUTH_OPEN，此字段将被忽略。</td>
</tr>
<tr>
<td>ssid_len</td>
<td>SSID 的长度。如果 ssid_len 为 0，则检查 SSID 直至出现终止字符。如果 ssid_len 大于 32，请更改为 32，或者根据 ssid_len 设置 SSID 长度。</td>
</tr>
<tr>
<td>channel</td>
<td>AP 的信道。如果信道超出范围，Wi-Fi 驱动程序将默认该信道为信道 1。所以，请确保信道在要求的范围内。有关详细信息，请参阅 Wi-Fi 协议/信道代码。</td>
</tr>
<tr>
<td>authmode</td>
<td>ESP AP 的身份验证模式。目前，ESP Wi-Fi 不支持 AUTH_WEP。如果 authmode 是一个无效值，AP 默认该值为 WIFI_AUTH_OPEN。</td>
</tr>
<tr>
<td>ssid_hidden</td>
<td>如果 ssid_hidden 为 1，AP 不广播 SSID，若为其他值，则广播。</td>
</tr>
<tr>
<td>max_connection</td>
<td>目前，ESP Wi-Fi 支持 10 个 Wi-Fi 连接。如果 max_connection 大于 10，AP 默认该值为 10。</td>
</tr>
<tr>
<td>beacon_interval</td>
<td>beacon 隔离。值为 100 ~ 60000 ms，默认值为 100 ms。如果该值不在上述范围，AP 默认取 100 ms。</td>
</tr>
</tbody>
</table>
Wi-Fi 协议模式

目前，IDF 支持以下协议模式：

<table>
<thead>
<tr>
<th>协议模式</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11b</td>
<td>调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B)，将 station/AP 设置为仅 802.11b 模式。</td>
</tr>
<tr>
<td>802.11bg</td>
<td>调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B</td>
</tr>
<tr>
<td>802.11g</td>
<td>调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B</td>
</tr>
<tr>
<td>802.11bgn</td>
<td>调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B</td>
</tr>
<tr>
<td>802.11gn</td>
<td>调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B</td>
</tr>
<tr>
<td>802.11 BGNLR</td>
<td>调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B</td>
</tr>
<tr>
<td>802.11 LR</td>
<td>调用函数 esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_LR) 和 esp_wifi_config_11b_rate(ifx, true)，将 station/AP 设置为 LR 模式。 此模式是乐鑫的专利模式，可以达到 1 公里视线范围。请确保 station 和 AP 同时连接至 ESP 设备。</td>
</tr>
</tbody>
</table>

长距离 (LR)

长距离 (LR) 模式是乐鑫的一项专利 Wi-Fi 模式，可达到 1 公里视线范围。与传统 802.11b 模式相比，接收灵敏度更高，抗干扰能力更强，传输距离更长。

**LR 兼容性** 由于 LR 是乐鑫的独有 Wi-Fi 模式，只有 ESP32 设备才能传输和接收 LR 数据。也就是说，如果连接的设备不支持 LR，ESP32 设备则不会以 LR 数据速率传输数据。可通过配置适当的 Wi-Fi 模式使您的应用程序实现这一功能。如果协商的模式支持 LR，ESP32 可能会以 LR 速率传输数据，否则，ESP32 将以传统 Wi-Fi 数据速率传输所有数据。

下表是 Wi-Fi 模式协商：

<table>
<thead>
<tr>
<th>APSTA</th>
<th>BGN</th>
<th>BG</th>
<th>B</th>
<th>BGNLR</th>
<th>BGLR</th>
<th>BLR</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGN</td>
<td>BGN</td>
<td>BG</td>
<td>B</td>
<td>BGN</td>
<td>BG</td>
<td>B</td>
<td>*</td>
</tr>
<tr>
<td>BG</td>
<td>BG</td>
<td>BG</td>
<td>B</td>
<td>BG</td>
<td>BG</td>
<td>B</td>
<td>*</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>*</td>
</tr>
<tr>
<td>BGNLR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>BGNLR</td>
<td>BGLR</td>
<td>BLR</td>
<td>LR</td>
</tr>
<tr>
<td>BGLR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>BGLR</td>
<td>BGLR</td>
<td>BLR</td>
<td>LR</td>
</tr>
<tr>
<td>BLR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>BLR</td>
<td>BLR</td>
<td>BLR</td>
<td>LR</td>
</tr>
<tr>
<td>LR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>LR</td>
<td>LR</td>
<td>LR</td>
<td>LR</td>
</tr>
</tbody>
</table>

上表中，行是 AP 的 Wi-Fi 模式，列是 station 的 Wi-Fi 模式。”-” 表示 AP 和 station 的 Wi-Fi 模式不兼容。 根据上表，得出以下结论：
对于已能使 LR 的 ESP32 AP，由于以 LR 模式发送 beacon，因此与传统的 802.11 模式不兼容。
对于已能使 LR 且并非仅 LR 模式的 ESP32 station，与传统 802.11 模式兼容。
如果 station 和 AP 都是 ESP32 设备，并且两者都使用 LR 模式，则协商的模式支持 LR。

如果协商的 Wi-Fi 模式同时支持传统的 802.11 模式和 LR 模式，则 Wi-Fi 驱动程序会在不同的 Wi-Fi 模式下自动选择最佳数据速率，应用程序无需任何操作。

**LR 对传统 Wi-Fi 设备的影响** 以 LR 速率进行的数据传输不会影响传统 Wi-Fi 设备。因为：
- LR 模式的 CCA 和回退过程符合 802.11 协议。
- 传统的 Wi-Fi 设备可以通过 CCA 检测 LR 信号并进行回退。

也就是说，LR 模式下传输效果与 802.11b 模式相似。

**LR 传输距离** LR 的接收灵敏度比传统的 802.11b 模式高 4 dB，理论上，传输距离约为 802.11b 的 2 至 2.5 倍。

**LR 吞吐量** 因为原始 PHY 数据传输速率为 1/2 Mbps 和 1/4 Mbps，LR 的吞吐量有限。

**何时使用 LR** 通常使用 LR 的场景包括：
- AP 和 station 都是乐鑫设备。
- 需要长距离 Wi-Fi 连接和数据传输。
- 数据吞吐量要求非常小，例如远程设备控制等。

**Wi-Fi 国家/地区代码**

调用 `esp_wifi_set_country()`，设置国家/地区信息。下表详细介绍了各个字段，请在配置这些字段之前参考当地的 2.4 GHz RF 操作规定。

<table>
<thead>
<tr>
<th>字段</th>
<th>描述</th>
</tr>
</thead>
</table>
- ASCII 码空格字符，代表 station/AP 所处国家/地区的规定允许当前频段所需的所有环境。  
- ASCII 码‘O’字符，代表 station/AP 所处国家/地区的规定仅允许室外环境。  
- ASCII 码‘I’字符，代表 station/AP 所处国家/地区的规定仅允许室内环境。  
- ASCII 码‘X’字符，代表 station/AP 所处国家/地区的规定允许在室外环境。  
- ASCII 码‘XX’字符，代表 station/AP 所处国家/地区的规定允许在室内环境。  
- ASCII 码 ‘O’字符，代表 station/AP 所处国家/地区的规定允许在室外环境。  |
| schan | 起始信道，station/AP 所处国家/地区规定的最小信道值。 |
| nchan | 规定的总信道数，比如，如果 schan=1, nchan=13, 那么 station/AP 可以从信道 1 至 13 发送数据。 |
| policy | 国家/地区策略。当配置的国家/地区信息与所选 AP 的国家/地区信息冲突时，该字段决定使用哪一信息。更多策略相关信息，可参见下文。 |

默认国家/地区信息为：

```c
wifi_country_t config = {  
  .cc = "01",  
  .schan = 1,  
  .nchan = 11,  
}  
```

(下页继续)
如果 Wi-Fi 模式为 station/AP 共存模式，则它们配置的国家/地区信息相同。有时，station 所连 AP 的国家/地区信息与配置的不同。例如，配置的 station 国家/地区信息为:

```c
wifi_country_t config = {
  .cc = "JP",
  .schan = 1,
  .nchan = 14,
  .policy = WIFI_COUNTRY_POLICY_AUTO,
};
```

但所连 AP 的国家/地区信息为:

```c
wifi_country_t config = {
  .cc = "CN",
  .schan = 1,
  .nchan = 13,
};
```

此时，使用所连 AP 的国家/地区信息。

下表描述了在不同 Wi-Fi 模式和不同国家/地区策略下使用的国家/地区信息，并描述了对主动扫描的影响。

<table>
<thead>
<tr>
<th>Wi-Fi 模式</th>
<th>策略</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>station 模式</td>
<td>WIFI_COUNTRY_POLICY_AUTO</td>
<td>AP 的 beacon 中有国家/地区的 IE，使用国家/地区信息为 beacon 中的信息，否则，使用默认信息。扫描时：主动扫描信道 1 至信道 11，被动扫描信道 12 至信道 14。请记住，如果带有隐藏 SSID 的 AP 和 station 被设置在被动扫描信道上，被动扫描将无法找到该 AP。也就是说，如果应用程序希望在每个信道中找到带有隐藏 SSID 的 AP，国家/地区信息应该配置为 WIFI_COUNTRY_POLICY_MANUAL。</td>
</tr>
<tr>
<td>station 模式</td>
<td>WIFI_COUNTRY_POLICY_MANUAL</td>
<td>扫描时：主动扫描信道 schan 至信道 schan+nchan-1。国家/地区信息。</td>
</tr>
<tr>
<td>AP 模式</td>
<td>WIFI_COUNTRY_POLICY_AUTO</td>
<td>配置的国家/地区信息。</td>
</tr>
<tr>
<td>AP 模式</td>
<td>WIFI_COUNTRY_POLICY_MANUAL</td>
<td>国家/地区信息。</td>
</tr>
<tr>
<td>station/AP 共存模式</td>
<td>WIFI_COUNTRY_POLICY_AUTO</td>
<td>station 模式与 station 模式下使用的国家/地区信息相同。如果 station 不连接任何外部 AP，AP 使用配置的国家/地区信息。如果 station 连接一个外部 AP，该 AP 的国家/地区信息与该 station 相同。</td>
</tr>
<tr>
<td>station/AP 共存模式</td>
<td>WIFI_COUNTRY_POLICY_MANUAL</td>
<td>station 模式与 station 模式下使用的国家/地区信息相同。该 AP 与 AP 模式、WIFI_COUNTRY_POLICY_MANUAL 策略下使用的国家/地区信息相同。</td>
</tr>
</tbody>
</table>

**主信道** AP 模式下，AP 的信道定义为主信道。station 模式下，station 所连 AP 的信道定义为主信道。station/AP 共存模式下，AP 和 station 的主信道必须相同。如果不同，station 的主信道始终优先。例如，初
Wi-Fi 供应商 IE 配置

默认情况下，所有 Wi-Fi 管理帧都由 Wi-Fi 驱动程序处理，应用程序不需要任何操作。但是，某些应用程序可能需要处理 beacon、probe request、probe response 和其他管理帧。例如，如果在管理帧中插入一些只针对供应商的 IE，则只有包含此 IE 的管理帧才能得到处理。ESP32 中，esp_wifi_set_vendor_ie() 和 esp_wifi_set_vendor_ie_cb() 负责此类任务。

4.33.15 Wi-Fi Easy Connect™ (DPP)

Wi-Fi Easy Connect™（也称为设备配置协议）是一个安全且标准化的配置协议，用于配置 Wi-Fi 设备。更多信息请参考 esp_dpp。

WPA2-Enterprise

WPA2-Enterprise 是企业无线网络的安全认证机制。在连接到接入点之前，它使用 RADIUS 服务器对网络用户进行身份验证。身份验证过程基于 802.1X 标准，并有不同的扩展身份验证协议 (EAP) 方法，如 TLS、TTLS、PEAP 等。RADIUS 服务器根据用户的凭据（用户名和密码）、数字证书或两者对用户进行身份验证。当处于 station 模式的 ESP32 尝试连接到企业模式的 AP 时，它会向 AP 发送身份验证请求。AP 将该请求发送到 RADIUS 服务器以对 station 进行身份验证。根据不同的 EAP 方法，可以通过 idf.py menuconfig 打开配置，并在配置中设置参数。ESP32 仅在 station 模式下支持 WPA2 Enterprise。

为了建立安全连接，AP 和 station 协商并使用最佳密码套件达成一致。ESP32 支持 AKM 的 802.1X/EAP (WPA) 方法和 AES-CCM（高级加密标准-密码块链消息验证码协议的计数器模式）支持的密码套件，设置了 USE_MBEDTLS_CRYPTO 标志，ESP32 也支持 mbedts 支持的密码套件。

目前，ESP32 支持以下 EAP 方法：
- EAP-TLS: 这是基于证书的方法，只需要 SSID 和 EAP-IDF。
- PEAP: PEAP: 这是受保护的 EAP 方法，用户名和密码是必填项。
- EAP-TTLS: 这是基于凭据的方法。只有服务器支持验证是强制性的，而用户身份验证是可选的。用户名和密码

- PAP: 密码认证协议
- CHAP: 询问握手身份验证协议
- MSCHAP 和 MSCHAP-V2
- EAP-FAST: 这是一种基于受保护的访问凭据 (PAC) 的认证方法，使用身份验证和密码。目前使用此功能时需要禁用 USE_MBEDTLS_CRYPTO 标志。

请查看 wifi/wifienterprise 获取关于证书创建以及如何在 ESP32 上运行 wpa2enterprise 示例的详细信息。

4.33.16 无线网络管理

无线网络管理让客户端设备能够交换有关网络拓扑结构的信息，包括与射频环境相关的信息。这使每个客户端都能感知到网络状况，从而促进无线网络性能的整体改进。这是 802.11v 规范的一部分。它还使客户端能够支持网络辅助漫游。网络辅助漫游让 WLAN 能够向关联的客户端发送消息，从而使客户端与具有更好链路指标的 AP 关联。这对于促进负载平衡以及引导连接不良的客户端很有用。

目前 802.11v 的实现支持 BSS 过渡管理帧。

4.33.17 无线资源管理

无线电资源测量（802.11k）旨在改善网络内流量的分配方式。在无线局域网中，一般情况下，无线设备会连接发射信号最强的接入点 (AP)。根据用户的数量和地理位置，这种分配方式有时会导致某个接入点超负荷而其它接入点利用不足，从而导致整体网络性能下降。在符合 802.11k 规范的网络中，如果信号
4.33.18  ESP32 Wi-Fi 节能模式

station 睡眠

目前，ESP32 Wi-Fi 支持 Modem-sleep 模式，该模式是 IEEE 802.11 协议中的传统节能模式。仅 station 模式支持该模式，station 必须先连接到 AP。如果使能了 Modem-sleep 模式，station 将定期在活动状态和睡眠状态之间切换。在睡眠状态下，RF、PHY 和 BB 处于关闭状态，以减少功耗。Modem-sleep 模式下，station 可以与 AP 保持连接。

Modem-sleep 模式包括最小和最大节能模式。在最小节能模式下，每个 DTIM 间隔，station 都将唤醒以接收 beacon，广播数据在 DTIM 之后传输，因此不会丢失。但是，由于 DTIM 间隔短由 AP 决定，如果该间隔时间设置较短，则省电效果不大。

在最大节能模式下，每个监听间隔，station 都将唤醒以接收 beacon。可以设置该监听间隔长于 AP 的 DTIM 周期。在 DTIM 间隔内，station 可能处于睡眠状态，广播数据会丢失。如果监听间隔较长，则可以节省更多电量，但广播数据更容易丢失。连接 AP 前，可以通过调用 API esp_wifi_set_config() 配置监听间隔。

调用 esp_wifi_init() 后，调用 esp_wifi_set_ps(WIFI_PS_MIN_MODEM) 可使能 Modem-sleep 最小节能模式。调用 esp_wifi_set_ps(WIFI_PS_MAX_MODEM) 可使能 Modem-sleep 最大节能模式。station 连接到 AP 时，Modem-sleep 模式将启动。station 与 AP 断开连接时，Modem-sleep 模式将停止。

调用 esp_wifi_set_ps(WIFI_PS_NONE) 可以完全禁用 Modem-sleep 模式。禁用会增大功耗，但可以最大限度减少实时接收 Wi-Fi 数据的延迟。使能 Modem-sleep 时，接收 Wi-Fi 数据的延迟时间可能与 DTIM 周期（最小节能模式）或监听间隔（最大节能模式）相同。在 Wi-Fi 与 Bluetooth LE 共存模式下，无法完全禁用 modem-sleep 模式。

默认的 Modem-sleep 模式是 WIFI_PS_MIN_MODEM。

AP 睡眠

目前，ESP32 AP 不支持 Wi-Fi 协议中定义的所有节能功能。具体来说，AP 只缓存所连 station 单播数据，不缓存组播数据。如果 ESP32 AP 所连的 station 已使能节能功能，可能发生组播数据包丢失。

未来，ESP32 AP 将支持所有节能功能。

非连接状态下的休眠

非连接状态指的是 esp_wifi_start() 至 esp_wifi_stop() 期间内，没有建立 Wi-Fi 连接的阶段。

目前，ESP32 Wi-Fi 支持以 station 模式运行时，在非连接状态 下休眠。可以通过选项 CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE 配置该功能。

如果打开配置选项 CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE，则在该阶段内，RF、PHY 和 BB 将在空闲时被关闭，电流将会等同于 Modem-sleep 模式下的休眠电流。

配置选项 CONFIG_ESP_WIFI_STA_DISCONNECTED_PM_ENABLE 默认情况下将会被打开，共存模式下被 Menuconfig 强制打开。
非连接模块功耗管理

非连接模块指的是一些不依赖于 Wi-Fi 连接的 Wi-Fi 模块，例如 ESP-NOW，DPP，FTM。这些模块从esp_wifi_start() 开始工作至esp_wifi_stop() 结束。

目前，ESP-NOW 以 station 模式工作时，既支持在连接状态下休眠，也支持在非连接状态下休眠。

非连接模块发包 对于任何非连接模块，在开启了休眠的任何时间点都可以发包，不需要进行任何额外的配置。

此外，esp_wifi_80211_tx() 也在休眠时被支持。

非连接模块收包 对于非连接模块，在开启休眠时如果需要进行收包，需要配置两个参数，分别为 Window 和 Interval。

在每个 Interval 开始时，RF, PHY 和 BB 将会被打开并保持 Window 的时间。非连接模块可以在此时间内收包。

Interval

• 全局只有一个 Interval 参数，所有非连接模块共享它。其数值由 API esp_wifi_set_connectionless_interval() 设置，单位为毫秒。
• Interval 的默认值为ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE。
• 在 Interval 开始时，将会给 WIFI_EVENT_CONNECTIONLESS_MODULE_WAKE_INTERVAL_START 事件，由于 Window 将在此时开始，可以在此事件内配置发包动作。
• 在连接状态下，Interval 开始的时间点将会与 TBTT 时间对齐。

Window

• 每个非连接模块在启动后都有其自身的 Window 参数，休眠模块将取所有模块 Window 的最大值运算。
• 其数值由 API module_name_set_wake_window() 设置，单位为毫秒。
• 模块 Window 的默认值为最大值。

<table>
<thead>
<tr>
<th>Interval</th>
<th>Esp_wifi_connectionless_interval_default_mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>0: not used</td>
</tr>
<tr>
<td></td>
<td>1 - maximum: default mode</td>
</tr>
<tr>
<td></td>
<td>used periodically (Window &lt; Interval) / used all time (Window = Interval)</td>
</tr>
</tbody>
</table>

默认模式 当 Interval 参数被配置为 ESP_WIFI_CONNECTIONLESS_INTERVAL_DEFAULT_MODE，且有非零的 Window 参数时，非连接模块功耗管理将会按照默认模式运行。

在没有与非 Wi-Fi 协议共存时，RF, PHY 和 BB 将会在默认模式下被一直打开。

在与非 Wi-Fi 协议共存时，RF, PHY 和 BB 资源被共存模块分时划给 Wi-Fi 非连接模块和非 Wi-Fi 协议使用。在默认模式下，Wi-Fi 非连接模块被允许周期性使用 RF, PHY 和 BB，并且具有稳定性能。

推荐在与非 Wi-Fi 协议共存时将非连接模块功耗管理配置为默认模式。

4.33.19 ESP32 Wi-Fi 吞吐量

下表是我们在 Espressif 实验室和屏蔽箱中获得的最佳吞吐量结果。
### 4.33.20 Wi-Fi 802.11 数据包发送

`esp_wifi_80211_tx()` API 可用于：

- 发送 beacon、probe request、probe response 和 action 帧。
- 发送非 QoS 数据帧。

不能用于发送加密或 QoS 帧。

#### 使用 `esp_wifi_80211_tx()` 的前提条件

- Wi-Fi 模式为 station 模式、AP 模式，或 station/AP 共存模式。
- API esp_wifi_set_promiscuous(true) 或 esp_wifi_start()，或者二者都返回 ESP_OK，这是为了确保在调用函数`esp_wifi_80211_tx()` 前，Wi-Fi 硬件已经初始化。对于 ESP32，esp_wifi_set_promiscuous(true) 和 esp_wifi_start() 都可以触发 Wi-Fi 硬件初始化。
- 提供正确的`esp_wifi_80211_tx()` 参数。

#### 传输速率

- 默认传输速率为 1 Mbps。
- 可以通过函数`esp_wifi_config_80211_tx_rate()` 设置任意速率。
- 可以通过函数`esp_wifi_set_bandwidth()` 设置任意带宽。

#### 在不同情况下需要避免的副作用

理论上，如果不考虑 API 对 Wi-Fi 驱动程序或其他 station 或 AP 的副作用，可以通过空中发送一个原始的 802.11 数据包，包括任何目的地址的 MAC、任何源地址的 MAC、任何 BSSID、或任何其他类型的数据包。但是，一个具有强健、有用的应用程序应该避免这种副作用。下表针对如何避免`esp_wifi_80211_tx()` 的副作用提供了一些提示或建议。

<table>
<thead>
<tr>
<th>类型/吞吐量</th>
<th>实验室空气状态</th>
<th>屏蔽箱</th>
<th>测试工具</th>
<th>IDF 版本 (commit ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>原始 802.11 数据包接收数据</td>
<td>N/A</td>
<td>130 MBit/s</td>
<td>内部工具</td>
<td>N/A</td>
</tr>
<tr>
<td>原始 802.11 数据包发送数据</td>
<td>N/A</td>
<td>130 MBit/s</td>
<td>内部工具</td>
<td>N/A</td>
</tr>
<tr>
<td>UDP 接收数据</td>
<td>30 MBit/s</td>
<td>85 MBit/s</td>
<td>iperf example</td>
<td>15575346</td>
</tr>
<tr>
<td>UDP 发送数据</td>
<td>30 MBit/s</td>
<td>75 MBit/s</td>
<td>iperf example</td>
<td>15575346</td>
</tr>
<tr>
<td>TCP 接收数据</td>
<td>20 MBit/s</td>
<td>65 MBit/s</td>
<td>iperf example</td>
<td>15575346</td>
</tr>
<tr>
<td>TCP 发送数据</td>
<td>20 MBit/s</td>
<td>75 MBit/s</td>
<td>iperf example</td>
<td>15575346</td>
</tr>
</tbody>
</table>

使用 iperf example 测试吞吐量时，sdkconfig 是 `examples/wifi/iperf/sdkconfig.defaults.esp32`。
### 场景  |  描述
--- | ---
无 Wi-Fi 连接 | 在这种情况下，因为没有 Wi-Fi 连接，Wi-Fi 驱动程序不会受到副作用影响。如果 en_sys_seq=true，则 Wi-Fi 驱动程序负责序列控制。如果 en_sys_seq=false，应用程序需要确保缓存区序列的正确。理论上，MAC 地址可以是任何地址，但是这样可能会影响其他使用相同 MAC/BSSID 的 station/AP。例如，AP 模式下，应用程序调用函数 esp_wifi_80211_tx() 发送带有 BSSID == mac_x 的 beacon，但是 mac_x 并非 AP 接口的 MAC。而且，还有另一个 AP（我们称之为“other-AP”）的 bssid 是 mac_x。因此，连接到“other-AP”的 station 无法分辨 beacon 来自“other-AP”还是 esp_wifi_80211_tx()，就会出现“意外行为”。为了避免上述副作用，我们建议：如果在 station 模式下调用函数 esp_wifi_80211_tx()，第一个 MAC 应该是组播 MAC 或是目标设备的 MAC，第二个 MAC 应该是 station 接口的 MAC。如果在 AP 模式下调用函数 esp_wifi_80211_tx()，第一个 MAC 应该是组播 MAC 或是目标设备的 MAC，第二个 MAC 应该是 AP 接口的 MAC。上述建议仅供参考，有充分理由的情况下可以忽略。
有 Wi-Fi 连接 | 当 Wi-Fi 已连接，且序列由应用程序控制，应用序程序可能会影响整个 Wi-Fi 连接的序列控制。因此，en_sys_seq 要为 true，否则将返回 ESP_ERR_WIFI_ARG。“无 Wi-Fi 连接”情况下的 MAC 地址选择也适用于此情况。如果 Wi-Fi 模式是 station 模式，MAC 的地址 1 是 station 所连 AP 的 MAC，地址 2 是 station 接口的 MAC。那么就称数据包是从 station 发送到 AP，另一方面，如果 Wi-Fi 模式是 AP 模式，且 MAC 地址 1 是该 AP 所连 station 的 MAC，地址 2 是 AP 接口的 MAC。那么就称数据包是从 AP 发送到 station。为避免与 Wi-Fi 连接冲突，可采用以下检查方法：如果数据包类型是数据，且是从 station 发送到 AP，IEEE 802.11 Frame control 字段中的 ToDS 位应该为 1，FromDS 位为 0，否则，Wi-Fi 驱动程序不接受该数据包。如果数据包类型是数据，且是从 AP 发送到 station，IEEE 802.11 Frame control 字段中的 ToDS 位应该为 0，FromDS 位为 1，否则，Wi-Fi 驱动程序不接受该数据包。如果数据包是从 station 发送到 AP，或从 AP 到 station，Power Management、More Data 和 Re-Transmission 位应该为 0，否则，Wi-Fi 驱动程序不接受该数据包。如果任何检查失败，将返回 ESP_ERR_WIFI_ARG。

#### 4.33.21 Wi-Fi Sniffer 模式

Wi-Fi Sniffer 模式可以通过 esp_wifi_set_promiscuous() 使能，如果使能 Sniffer 模式，可以向应用程序转储以下数据包。

- 802.11 管理帧
- 802.11 数据帧，包括 MPDU、AMPDU、AMSDU 等
- 802.11 MIMO 帧，Sniffer 模式仅转储 MIMO 帧的长度。
- 802.11 控制帧
- 802.11 CRC 错误帧

不可以向应用程序转储以下数据包。

- 802.11 其它错误帧

对于 Sniffer 模式可以转储的帧，应用程序可以另外使用 esp_wifi_set_promiscuous_filter() 和 esp_wifi_set_promiscuous_ctrl_filter() 决定筛选哪些特定类型的数据包。应用程序默认筛选所有 802.11 数据和管理帧。

可以在 WIFI_MODE_NULL、WIFI_MODE_STA、WIFI_MODE_AP、WIFI_MODE_APSTA 等 Wi-Fi 模式下使能 Wi-Fi Sniffer 模式，也就是说，当 station 连接到 AP，或者 AP 有 Wi-Fi 连接时，就可以使能。请注意，Sniffer 模式对 station/AP Wi-Fi 连接的吞吐量有很大影响。通常，除非有特别原因，当 station/AP Wi-Fi 连接出现大量流量，不应使能。

该模式下还应注意回调函数 wifi_promiscuous_cb 的使用。该回调将直接在 Wi-Fi 驱动程序任务中进行。
所以如果应用程序需处理大量过滤的数据包，建议在回调中向应用程序任务发布一个事件，把真正的工作推延到应用程序任务中完成。

### 4.33.22 Wi-Fi 多根天线

下图描述 Wi-Fi 多根天线的选择过程:

```plaintext
<table>
<thead>
<tr>
<th>Enabled</th>
<th>Antenna 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

ESP32 通过外部天线开关，最多支持 16 根天线。天线开关最多可由四个地址管脚控制 - antenna_select[0:3]，向 antenna_select[0:3] 输入不同的值，以选择不同的天线。例如，输入值 ‘0b1011’ 表示选中天线 11。antenna_select[3:0] 的默认值为 “00000”，表示默认选择了天线 0。

四个高电平有效 antenna_select 管脚有多达四个 GPIO 连接。ESP32 可以通过控制 GPIO[0:3] 选择天线。API esp_wifi_set_ant_gpio() 用于配置 antenna_select 连接哪些 GPIO。如果 GPIO[x] 连接到 antenna_select[x], gpio_config->gpio_cfg[x].gpio_select 应设置为 1，其导提供 gpio_config->gpio_cfg[x].gpio_num 的值。

天线开关的具体实现不同，antenna_select[0:3] 的输入值中可能存在非法值，即 ESP32 通过外部天线开关支持的天线数可能小于 16 根。例如，ESP32-WROOM-DA 使用 RTC6603P 作为天线开关，仅支持 2 根天线。两个天线选择输入管脚为高电平有效，连接到两个 GPIO。’0b11’ 表示选中天线 0，’0b01’ 表示选中天线 1，输入值 ‘0b00’ 和 ‘0b11’ 为非法值。

尽管最多支持 16 根天线，发送和接收数据时，最多只能同时使用两根天线。API esp_wifi_set_ant() 用于配置使能哪些天线。

使能天线后，选择算法的过程同样可由 esp_wifi_set_ant() 配置。接收/发送数据的天线模式可以是 WIFI_ANT_MODE_ANT0、WIFI_ANT_MODE_ANT1 或 WIFI_ANT_MODE_AUTO。如果天线模式为 WIFI_ANT_MODE_ANT0，使能的天线 0 用于接收/发送数据。如果天线模式为 WIFI_ANT_MODE_ANT1，使能的天线 1 用于接收/发送数据。否则，Wi-Fi 会自动选择使能天线中信号较好的天线。

如果接收数据的天线模式为 WIFI_ANT_MODE_AUTO，还需要设置默认天线模式。只有在满足某些条件时，接收数据天线才切换，例如，如果 RSSI 低于 -65 dBm，或另一根天线信号更好。如果条件不满足，接收数据使用默认天线。如果默认天线模式为 WIFI_ANT_MODE_ANT1，则使能的天线 1 是默认接收数据天线，否则是使能的天线 0。

有一些限制情况需要考虑:

- 因为发送数据天线基于 WIFI_ANT_MODE_AUTO 类型的接收数据天线选择算法，只有接收数据的天线模式为 WIFI_ANT_MODE_AUTO 时，发送数据天线才能设置为 WIFI_ANT_MODE_AUTO。
- 目前，Bluetooth® 不支持多根天线功能，请不要使用与多根天线有关的 API。

推荐在以下场景中使用多根天线:

- Wi-Fi 模式 WIFI_MODE_STA 下，接收/发送数据的天线模式均配置为 WIFI_ANT_MODE_AUTO。Wi-Fi 驱动程序自动选择更好的接收/发送天线。
- 接收数据天线模式配置为 WIFI_ANT_MODE_AUTO, 发送数据的天线模式配置为 WIFI_ANT_MODE_ANT1 或 WIFI_ANT_MODE_ANT1，应用程序可以始终选择指定的天线用于发送数据，也可以执行自身发送数据天线选择算法，如根据信号切换信息选择发送数据的天线模式等。
• 接收/发送数据的天线模式均配置为 WIFI_ANT_MODE_ANT0 或 WIFI_ANT_MODE_ANT1。

**Wi-Fi 多根天线配置**

通常，可以执行以下步骤来配置多根天线：

- 配置 antenna_selects 连接哪些 GPIOs，例如，如果支持四根天线，且 GPIO20/GPIO21 连接到 antenna[0]/antenna_select[1]，配置如下所示：

```c
wifi_ant_gpio_config_t config = {
    .gpio_select = 1, .gpio_num = 20,
    .gpio_select = 1, .gpio_num = 21
};
```

- 配置使能哪些天线，以及接收/发送数据如何使用使能的天线，例如，如果使能了天线 1 和天线 3，接收数据需要自动选择较好的天线，并将天线 1 作为默认天线，发送数据最终选择天线 3。配置如下所示：

```c
wifi_ant_config_t config = {
    .rx_ant_mode = WIFI_ANT_MODE_AUTO,
    .rx_ant_default = WIFI_ANT_ANT0,
    .tx_ant_mode = WIFI_ANT_MODE_ANT1,
    enabled_ant0 = 1,
    enabled_ant1 = 3
};
```

### 4.33.23 Wi-Fi 信道状态信息

信道状态信息 (CSI) 是指 Wi-Fi 连接的信道信息。ESP32 中，CSI 由子载波的信道频率响应组成。CSI 从发送端接收数据包时开始估计，每个子载波信道频率响应由两个字节的有符号字符记录。第一个字节是虚部，第二个字节是实部。根据接收数据包的类型，信道频率响应最多有三个字段。分别是 LLTF、HT-LTF 和 STBC-HT-LTF。对于在不同状态的信道上接收到的不同类型的信包、CSI 的子载波索引和总字节数如下表所示。


<table>
<thead>
<tr>
<th>信道</th>
<th>辅助信道</th>
<th>数据包信息</th>
<th>下</th>
<th>上</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>非 STBC</td>
<td>非 STBC</td>
<td>HT</td>
<td>非 HT</td>
</tr>
<tr>
<td>信道</td>
<td>非 STBC</td>
<td>非 STBC</td>
<td>非 STBC</td>
<td>非 STBC</td>
</tr>
<tr>
<td>带宽</td>
<td>20 MHz</td>
<td>20 MHz</td>
<td>20 MHz</td>
<td>20 MHz</td>
</tr>
<tr>
<td></td>
<td>0~31</td>
<td>0~31</td>
<td>0~31</td>
<td>0~31</td>
</tr>
<tr>
<td></td>
<td>0<del>32</del>1</td>
<td>0<del>32</del>1</td>
<td>0<del>32</del>1</td>
<td>0<del>32</del>1</td>
</tr>
<tr>
<td></td>
<td>0~63</td>
<td>0~63</td>
<td>0~63</td>
<td>0~63</td>
</tr>
<tr>
<td></td>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
</tr>
<tr>
<td></td>
<td>0~62</td>
<td>0~62</td>
<td>0~62</td>
<td>0~62</td>
</tr>
<tr>
<td></td>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
</tr>
<tr>
<td>子载波索引</td>
<td>LLTF</td>
<td>HT-LTF</td>
<td>STBC-HT-LTF</td>
<td></td>
</tr>
<tr>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td>0<del>31, 32</del>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td>0<del>60</del>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

表中的所有信息可以在 wifi_csi_info_t 结构中找到。

- 辅助信道指 rx_ctrl 字段的 secondary_channel 字段。
- 数据包的信号模式指 rx_ctrl 字段的 sig_mode 字段。
- 信道带宽指 rx_ctrl 字段中的 cwb 字段。
子载波的虚部和实部的使用请参考下表。

<table>
<thead>
<tr>
<th>PHY 标准</th>
<th>子载波范围</th>
<th>导频子载波</th>
<th>子载波个数 (总数/数据子载波)</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11a/g</td>
<td>-26到+26</td>
<td>-21,-7,+7,+21</td>
<td>52 total, 48 usable</td>
</tr>
<tr>
<td>802.11n, 20MHz</td>
<td>-28到+28</td>
<td>-21,-7,+7,+21</td>
<td>56 total, 52 usable</td>
</tr>
<tr>
<td>802.11n, 40MHz</td>
<td>-57到+57</td>
<td>-53,-25,-11,+11,+25,+53</td>
<td>114 total, 108 usable</td>
</tr>
</tbody>
</table>

备注：
- 对于 STBC 数据包，每个时隙都提供了 CSI，不会出现 CSD（循环移位延迟）。由于附带链上的
  每一次循环移位为 -200 ns，因为子载波 0 中没有信道频率响应，在 HT-LTF 和 STBC-HT-LTF
  中只记录第一时隙的 CSD 角度。CSD[10] 是 11 位，范围从 -pi 到 pi。
- 如果调用 API esp_wifi_set_csi_config() 没有使能 LLTF、HT-LTF 或 STBC-HT-LTF，则 CSI
  数据的总字节数量会比表中的少。例如，如果没有使能 LLTF 和 HT-LTF，而使能 STBC-HT-LTF，当
  接收到上述条件、HT、40 MHz 或 STBC 的数据包时，CSI 数据的总字节数为 244 ((61+60)*2+2=244，
  结果对齐为四个字节，最后两个字节无效)。

### 4.33.24 Wi-Fi 信道状态信息配置

要使用 Wi-Fi CSI，需要执行以下步骤。
- 在菜单配置中选择 Wi-Fi CSI，方法是“菜单配置 -> 组件配置 -> Wi-Fi -> Wi-Fi CSI (信道状态信息)”。
- 调用 API esp_wifi_set_csi_rx_cb() 设置 CSI 接收回调函数。
- 调用 API esp_wifi_set_csi_config() 配置 CSI。
- 调用 API esp_wifi_set_csi() 使能 CSI。

CSI 接收回调函数从 Wi-Fi 任务中运行。因此，不要在回调函数中进行冗长的操作。可以将需要的数据
发送到队列中，并从一个较低优先级的任务中处理。由于 station 在断开连接时不会收到任何数据包，只
有在连接时才会收到来自 AP 的数据包，因此建议通过调用函数 esp_wifi_set_promiscuous() 使
能 Sniffer 模式接收更多 CSI 数据。

### 4.33.25 Wi-Fi HT20/40

ESP32 支持 Wi-Fi 带宽 HT20 或 HT40，不支持 HT20/40 共存。调用函数 esp_wifi_set_bandwidth() 可改变
station/AP 的默认带宽。ESP32 station 和 AP 的默认带宽为 HT40。

station 模式下，实际带宽首先在 Wi-Fi 连接时协商。只有当 station 和所连 AP 都支持 HT40 时，带宽才为
HT40，否则为 HT20。如果所连的 AP 的带宽发生变化，则在不断开 Wi-Fi 连接的情况下再次协商实际带
宽。

同样，在 AP 模式下，在 AP 与所连 station 协商实际带宽。如果 AP 和其中一个 station 支持 HT40，则为
HT40，否则为 HT20。

在 station/AP 共存模式下，station 和 AP 都可独立配置为 HT20/40。如果 station 和 AP 都协商为 HT40，由
于 ESP32 中，station 的优先级总高于 AP，HT40 信道是 station 的信道。例如，AP 的配置带宽为 HT40，
配置的主信道为 6，配置的辅助信道为 10。如果，station 所连路由器的主信道为 6，辅助信道为 2，AP 的
实际信道将自动更改为主 6 和辅 2。
理论上，HT40 可以获得更大的吞吐量。因为 HT40 的最大原始 PHY 数据速率为 150 Mbps，而 HT20 为 72 Mbps。但是，如果设备在某些特殊环境中使用，例如，ESP32 周围其他 Wi-Fi 设备过多，HT40 的性能可能会降低。因此，如果应用程序需要支持相同或类似的情况，建议始终将带宽配置为 HT20。

4.33.26 Wi-Fi QoS

ESP32 支持 WFA Wi-Fi QoS 认证所要求的所有必备功能。

Wi-Fi 协议中定义了四个 AC（访问类别），每个 AC 有各自的优先级访问 Wi-Fi 信道。此外，还定义了映射规则以映射其他协议的 QoS 优先级，例如 802.11D 或 TCP/IP 到 Wi-Fi AC。

下表描述 ESP32 中 IP 优先级如何映射到 Wi-Fi AC，还指明此 AC 是否支持 AMPDU。该表按优先级降序排列，即 AC_VO 拥有最高优先级。

<table>
<thead>
<tr>
<th>IP 优先级</th>
<th>Wi-Fi AC</th>
<th>是否支持 AMPDU</th>
</tr>
</thead>
<tbody>
<tr>
<td>6, 7</td>
<td>AC_VO (Voice)</td>
<td>否</td>
</tr>
<tr>
<td>4, 5</td>
<td>AC_VI (Video)</td>
<td>是</td>
</tr>
<tr>
<td>3, 0</td>
<td>AC_BE (Best Effort)</td>
<td>是</td>
</tr>
<tr>
<td>1, 2</td>
<td>AC_BK (Background)</td>
<td>是</td>
</tr>
</tbody>
</table>

应用程序可以通过套接字选项 IP_TOS 配置 IP 优先级使用 QoS 功能。下面是使套接字使用 VI 队列的示例:

```c
const int ip_precedence_vii = 4;
const int ip_precedence_offset = 5;
int priority = (ip_precedence_vii << ip_precedence_offset);
setsockopt(socket_id, IPPROTO_IP, IP_TOS, &priority, sizeof(priority));
```

理论上，高优先级的 AC 比低优先级 AC 具有更好的性能，但并非总是如此，下面是一些关于如何使用 Wi-Fi QoS 的建议:

- 可以把一些真正重要的应用程序流量放到 AC_VO 队列中。避免通过 AC_VO 队列发送大流量。一方面，AC_VO 队列不支持 AMPDU，如果流量很大，性能不会优于其他队列。另一方面，可能会严重影响同样使用 AC_VO 队列的管理帧。
- 避免使用 AMPDU 支持的，两个以上的不同优先级，比如 socket A 使用优先级 0，socket B 使用优先级 1，socket C 使用优先级 2。因为可能需要更多的内存，不是好的设计。具体来说，Wi-Fi 驱动程序可能会为每个优先级生成一个 Block Ack 会话，如果设置了 Block Ack 会话，则需要更多内存。

4.33.27 Wi-Fi AMSDU

ESP32 支持接收和发送 AMSDU。

4.33.28 Wi-Fi 分片

支持 Wi-Fi 接收分片，但不支持 Wi-Fi 发送分片。

4.33.29 WPS 注册

在 Wi-Fi 模式 WIFI_MODE_STA 或 WIFI_MODE_APSTA 下，ESP32 支持 WPS 注册功能。目前，ESP32 支持的 WPS enrollee 类型有 PBC 和 PIN。

4.33.30 Wi-Fi 缓冲区使用情况

本节只介绍动态缓冲区配置。
Chapter 4. API 指南

缓冲区配置的重要性

为了获得一个具有强健、高性能的系统，我们需要非常谨慎地考虑内存的使用或配置情况，因为：

- ESP32 的可用内存有限。
- 目前，LwIP 和 Wi-Fi 驱动程序中默认的缓冲区类型是“动态”，意味着 LwIP 和 Wi-Fi 都与应用程序共享内存。程序员应该时刻牢记这一点，否则将面临如“堆内存耗尽”等的内存问题。
- “堆耗尽”情况非常危险，会导致 ESP32 出现“未定义行为”。因此，应该为应用程序预留足够的堆内存，防止耗尽。
- Wi-Fi 的吞吐量很大程度上取决于与内存相关的配置，如 TCP 窗口大小、Wi-Fi 接收/发送数据动态缓冲区数量等。
- ESP32 LwIP/Wi-Fi 可能使用的堆内存峰值取决于许多因素，例如应用程序可能拥有的最大 TCP/UDP 连接等。
- 在考虑内存配置时，应用程序所需的总内存也是一个重要因素。

由于这些原因，不存在一个适合所有应用程序的配置。相反，我们必须为每个不同的应用程序考虑不同的内存配置。

动态与静态缓冲区

Wi-Fi 驱动程序中默认的缓存类型是 “动态”。大多数情况下，动态缓冲区可以极大地节省内存。但是因为应用程序需要考虑 Wi-Fi 的内存使用情况，会给应用程序编程造成一定的难度。

LwIP 还在 TCP/IP 层分配缓冲区，这种缓冲区分配也是动态的。具体内容，见 LwIP 文档内存使用和性能部分。

Wi-Fi 动态缓冲区峰值

Wi-Fi 驱动程序支持多种类型的缓冲区（参考 Wi-Fi 缓冲区配置）。但本节只介绍 Wi-Fi 动态缓冲的使用方法 Wi-Fi 使用的堆内存峰值是 Wi-Fi 驱动程序理论上消耗的最大内存。通常，该内存峰值取决于：

- 配置的动态接收数据缓冲区数：wifi_rx_dynamic_buf_num
- 配置的动态发送数据缓冲区数：wifi_tx_dynamic_buf_num
- Wi-Fi 驱动程序可以接收的最大数据包：wifi_rx_pkt_size_max
- Wi-Fi 驱动程序可以发送的最大数据包：wifi_tx_pkt_size_max

因此，Wi-Fi 驱动程序消耗的内存峰值可以用下面的公式计算：

$$\text{wifi\_dynamic\_peek\_memory} = (\text{wifi\_rx\_dynamic\_buf\_num} \times \text{wifi\_rx\_pkt\_size\_max}) + (\text{wifi\_tx\_dynamic\_buf\_num} \times \text{wifi\_tx\_pkt\_size\_max})$$

一般情况下，不需要关心动态发送数据长缓冲区和超长缓冲区，因为它们是管理帧，对系统的影响很小。

4.3.31 如何提高 Wi-Fi 性能

ESP32 Wi-Fi 的性能受许多参数的影响，各参数之间存在相互制约。如果配置得合理，不仅可以提高性能，还可以增加应用程序的可用内存，提高稳定性。

在本节中，我们将简单介绍 Wi-Fi/LWIP 协议栈的工作模式，并说明各个参数的作用。我们将推荐几种配置等级，您可以根据使用场景选择合适的等级。

协议栈工作模式

ESP32 协议栈分为四层，分别为应用层、LWIP 层、Wi-Fi 层和硬件层。

- 在接收过程中，硬件将接收到的数据包放入 DMA 缓冲区，然后依次传送到 Wi-Fi 的接收数据缓冲区、LWIP 的接收数据缓冲区进行相关协议处理，最后传送到应用层。Wi-Fi 的接收数据缓冲区和 LWIP 的接收数据缓冲区默认共享同一个缓冲区。也就是说，Wi-Fi 默认将数据包转发到 LWIP 作为参考。
在发送过程中，应用程序首先将要发送的消息复制到LWIP层的发送数据缓冲区，进行TCP/IP封装。然后将消息发送到Wi-Fi层的发送数据缓冲区进行MAC封装，最后等待发送。

### 参数

适当增加上述缓冲区的大小或数量，可以提高Wi-Fi性能，但同时会减少应用程序的可用内存。下面我们将介绍您需要配置的参数：

#### 接收数据方向：

- **CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM** 该参数表示硬件层的DMA缓冲区数量。提高该参数将增加发送方的一次性接收吞吐量，从而提高Wi-Fi协议栈处理突发流量的能力。
- **CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM** 该参数表示Wi-Fi层中接收数据缓冲区的数量。提高该参数可以增强数据包的接收性能。该参数需要与LWIP层的接收数据缓冲区大小相匹配。
- **CONFIG_ESP32_WIFI_RX_BA_WIN** 该参数表示接收端AMPU BA窗口的大小。应配置为**CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM**和**CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM**的二倍数值中较小的数值。
- **CONFIG_LWIP_TCP_WND_DEFAULT** 该参数表示LWIP层用于每个TCP流的接收数据缓冲区大小，应配置为**WIFI_DYNAMIC_RX_BUFFER_NUM**（KB）的值，从而实现高稳定性。同时，在有多个流的情况下，应相应降低该参数值。

#### 发送数据方向：

- **CONFIG_ESP32_WIFI_TX_BUFFER** 该参数表示发送数据缓存区的类型，建议配置为动态缓存区，该配置可以充分利用内存。
- **CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM** 该参数表示Wi-Fi层发送数据缓冲区数量。提高该参数可以增强数据包发送的性能。该参数值需要与LWIP层的发送数据缓冲区大小相匹配。
- **CONFIG_LWIP_TCP_SND_BUF_DEFAULT** 该参数表示LWIP层用于每个TCP流的发送数据缓冲区大小，应配置为**WIFI_DYNAMIC_TX_BUFFER_NUM**（KB）的值，从而实现高稳定性。在有多个流的情况下，应相应降低该参数值。

#### 通过在IRAM中放置代码优化吞吐量：

- **CONFIG_ESP32_WIFI_IRAM_OPT** 如果能使该选项，一些Wi-Fi功能将被移至IRAM，从而提高吞吐量，IRAM使用量将增加15 KB。
- **CONFIG_ESP32_WIFI_RX_IRAM_OPT** 如果能使该选项，一些Wi-Fi接收数据功能将被移至IRAM，从而提高吞吐量，IRAM使用量将增加16 KB。
• **CONFIG_LWIP_IRAM_OPTIMIZATION** 如果使能该选项，一些 LWIP 功能将被移至 IRAM，从而提高吞吐量。IRAM 使用量将增加 13 KB。

备注：上述的缓冲区大小固定为 1.6 KB。

### 如何配置参数

ESP32 的内存由协议栈和应用程序共享。

在这里，我们给出了几种配置等级。在大多数情况下，您应根据应用程序所占用内存的大小，选择合适的等级进行参数配置。

下表中未提及的参数应设置为默认值。

<table>
<thead>
<tr>
<th>参数</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>可用内存 (KB)</td>
<td>37.1</td>
<td>113.8</td>
<td>123.3</td>
<td>145.5</td>
<td>144.5</td>
<td>170.2</td>
<td>185.2</td>
</tr>
<tr>
<td>WIFI_STATIC_RX_BUF_NUM</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WIFI_DYNAMIC_RX_BUF_NUM</td>
<td>124</td>
<td>34</td>
<td>20</td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WIFI_DYNAMIC_TX_BUF_NUM</td>
<td>24</td>
<td>18</td>
<td>20</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>WIFI_RX_BA_WIN</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>10</td>
<td>6</td>
<td>禁用</td>
</tr>
<tr>
<td>TCP_SND_BUF_MAX (KB)</td>
<td>28</td>
<td>24</td>
<td>18</td>
<td>20</td>
<td>12</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>TCP_WND_MAX (KB)</td>
<td>16</td>
<td>24</td>
<td>34</td>
<td>20</td>
<td>12</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WIFI_IRAM_OPT</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>WIFI_RX_IRAM OPT</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>LWIP_IRAM OPT</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>TCP 发送数据吞吐量 (Mbit/s)</td>
<td>74.6</td>
<td>50.8</td>
<td>46.5</td>
<td>39.9</td>
<td>44.2</td>
<td>33.8</td>
<td>25.6</td>
</tr>
<tr>
<td>TCP 接收数据吞吐量 (Mbit/s)</td>
<td>63.6</td>
<td>35.5</td>
<td>42.3</td>
<td>48.5</td>
<td>40.5</td>
<td>30.1</td>
<td>27.8</td>
</tr>
<tr>
<td>UDP 发送数据吞吐量 (Mbit/s)</td>
<td>76.2</td>
<td>75.1</td>
<td>74.1</td>
<td>72.4</td>
<td>69.6</td>
<td>64.1</td>
<td>36.5</td>
</tr>
<tr>
<td>UDP 接收数据吞吐量 (Mbit/s)</td>
<td>83.1</td>
<td>66.3</td>
<td>75.1</td>
<td>75.6</td>
<td>73.1</td>
<td>65.3</td>
<td>54.7</td>
</tr>
</tbody>
</table>

备注：以上结果使用华硕 RT-N66U 路由器，在屏蔽箱中进行单流测试得出。ESP32 的 CPU 为双核，频率为 240 MHz，flash 为 QIO 模式，频率为 80 MHz。

等级：

- **Iperf 级别** ESP32 的极端性能等级，用于测试极端性能。
- **高性能级别** ESP32 的高性能配置等级，适用于应用程序占用内存较少但有高性能要求的场合。在该等级中，您可以根据使用场景选择使用接收数据优先等级或发送数据优先等级。
- **默认级别** ESP32 的默认配置等级，兼顾可用内存和性能。
- **节省内存级别** 该级别适用于应用程序需要大量内存的场景。在这一等级下，收发器的性能会有所降低。
- **最小级别** ESP32 的最小配置等级。协议栈只使用运行所需的内存。适用于对性能没有要求，而应用程序需要大量内存的场景。
使用 PSRAM

PSRAM 一般在应用程序占用大量内存时使用。在该模式下，CONFIG_ESP32_WIFI_TX_BUFFER 被强制为静态。CONFIG_ESP32_WIFI_STATIC_TX_BUFFER_NUM 表示硬件层 DMA 缓冲区数量，提高这一参数可以增强性能。以下是使用 PSRAM 时的推荐等级。

<table>
<thead>
<tr>
<th>等级</th>
<th>lperf</th>
<th>默认</th>
<th>节省内存</th>
<th>最小</th>
</tr>
</thead>
<tbody>
<tr>
<td>可用内存 (KB)</td>
<td>113.8</td>
<td>152.4</td>
<td>181.2</td>
<td>202.6</td>
</tr>
<tr>
<td>WIFI_STATIC_RX_BUFFER_NUM</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>WIFI_DYNAMIC_RX_BUFFER_NUM</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>WIFI_STATIC_TX_BUFFER_NUM</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>WIFI_RX_BW</td>
<td>16</td>
<td>8</td>
<td>禁用</td>
<td></td>
</tr>
<tr>
<td>TCP_SND_BUF_DEFAULT (KB)</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>TCP_WND_DEFAULT (KB)</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>WIFI_IRAM_OPT</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>WIFI_RX_IRAM_OPT</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>LWIP_IRAM_OPTIMIZATION</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>TCP 发送数据吞吐量 (Mbit/s)</td>
<td>37.5</td>
<td>31.7</td>
<td>21.7</td>
<td>14.6</td>
</tr>
<tr>
<td>TCP 接收数据吞吐量 (Mbit/s)</td>
<td>31.5</td>
<td>29.8</td>
<td>26.5</td>
<td>21.1</td>
</tr>
<tr>
<td>UDP 发送数据吞吐量 (Mbit/s)</td>
<td>69.1</td>
<td>31.5</td>
<td>27.1</td>
<td>24.1</td>
</tr>
<tr>
<td>UDP 接收数据吞吐量 (Mbit/s)</td>
<td>40.1</td>
<td>38.5</td>
<td>37.5</td>
<td>36.9</td>
</tr>
</tbody>
</table>

4.33.32 Wi-Fi Menuconfig

Wi-Fi 缓冲区配置

如果您要修改默认的缓冲区数量或类型，最好也了解缓冲区在数据路径中是如何分配或释放的。下图显示了发送数据方向的这一过程。

- 应用程序分配需要发送的数据。
- 应用程序调用 TCP/IP 或套接字相关的 API 发送用户数据。这些 API 会分配一个在 LwIP 中使用的 PBUF，并复制用户数据。
- 当 LwIP 调用 Wi-Fi API 发送 PBUF 时，Wi-Fi API 会分配一个“动态发送数据缓冲区”或“静态发送数据缓冲区”，并复制 LwIP PBUF，最后发送数据。

下图展示了如何在接收数据方向分配或释放缓冲区：

图 62: TX Buffer Allocation

描述：

- 应用程序分配需要发送的数据。
- 应用程序调用 TCP/IP 或套接字相关的 API 发送用户数据。这些 API 会分配一个在 LwIP 中使用的 PBUF，并复制用户数据。
- 当 LwIP 调用 Wi-Fi API 发送 PBUF 时，Wi-Fi API 会分配一个“动态发送数据缓冲区”或“静态发送数据缓冲区”，并复制 LwIP PBUF，最后发送数据。
图 63: 接收数据缓冲区分配

描述:

- Wi-Fi 硬件在空中接收到数据包后，将数据包内容放到“静态接收数据缓冲区”，也就是“接收数据 DMA 缓冲区”。
- Wi-Fi 驱动程序分配一个“动态接收数据缓冲区”，复制“静态接收数据缓冲区”，并将“静态接收数据缓冲区”返回给硬件。
- Wi-Fi 驱动程序将数据包传送到上层 (LwIP)，并分配一个 PBUF 用于存放“动态接收数据缓冲区”。
- 应用程序从 LwIP 接收数据。

下表是 Wi-Fi 内部缓冲区的配置情况。
<table>
<thead>
<tr>
<th>缓冲区类型</th>
<th>分配类型</th>
<th>默认</th>
<th>是否可配置</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>静态接收数据缓冲区（硬件接收数据缓冲区）</td>
<td>静态</td>
<td>10 * 1600 Bytes</td>
<td>是</td>
<td>这是一种 DMA 内存，在函数 esp_wifi_init() 中初始化，在函数 esp_wifi_deinit() 中释放。该缓冲区形成硬件接收列表。当通过空中接收到一个帧时，硬件将该帧写入缓冲区，并向 CPU 发起一个中断。然后，Wi-Fi 驱动程序从缓冲区中读取内容，并将缓冲区返回到列表中。如果应用程序希望减少 Wi-Fi 静态分配的内存，可以将该值从 10 减少到 6，从而节省 6400 Bytes 的内存。除非禁用 AMPDU 功能，否则不建议将该值降低到 6 以下。</td>
</tr>
<tr>
<td>动态接收数据缓冲区</td>
<td>动态</td>
<td>32</td>
<td>是</td>
<td>缓冲区的长度可变，取决于所接收帧的长度。当 Wi-Fi 驱动程序从“硬件接收数据缓冲区”接收到一帧时，需要从堆中分配“动态接收数据缓冲区”。在 Menuconfig 中配置的“动态接收数据缓冲区”数量用来限制未释放的“动态接收数据缓冲区”总数量。</td>
</tr>
<tr>
<td>动态发送数据缓冲区</td>
<td>动态</td>
<td>32</td>
<td>是</td>
<td>这是一种 DMA 内存，位于堆内存中。当上层（LwIP）向 Wi-Fi 驱动程序发送数据包时，该缓冲区首先分配一个“动态发送数据缓冲区”，并复制上层缓冲区。动态发送数据缓冲区和静态发送数据缓冲区相互排斥。</td>
</tr>
<tr>
<td>静态发送数据缓冲区</td>
<td>静态</td>
<td>16 * 1600 Bytes</td>
<td>是</td>
<td>这是一种 DMA 内存，在函数 esp_wifi_init() 中初始化，在函数 esp_wifi_deinit() 中释放。当上层（LwIP）向 Wi-Fi 驱动程序发送数据包时，该缓冲区首先分配一个“静态发送数据缓冲区”，并复制上层缓冲区。动态发送数据缓冲区和静态发送数据缓冲区相互排斥。由于发送数据缓冲区必须是 DMA 缓冲区，所以当使用 PSRAM 时，发送数据缓冲区必须是静态的。</td>
</tr>
<tr>
<td>管理短缓冲区</td>
<td>动态</td>
<td>8</td>
<td>否</td>
<td>Wi-Fi 驱动程序的内部缓冲区。</td>
</tr>
<tr>
<td>管理长缓冲区</td>
<td>动态</td>
<td>32</td>
<td>否</td>
<td>Wi-Fi 驱动程序的内部缓冲区。</td>
</tr>
<tr>
<td>管理超长缓冲区</td>
<td>动态</td>
<td>32</td>
<td>否</td>
<td>Wi-Fi 驱动程序的内部缓冲区。</td>
</tr>
</tbody>
</table>

**Wi-Fi NVS Flash**

如果使能 Wi-Fi NVS flash，所有通过 Wi-Fi API 设置的 Wi-Fi 配置都会被存储到 flash 中，Wi-Fi 驱动程序在下次开机或重启时将自动加载这些配置。但是，应用程序可视情况禁用 Wi-Fi NVS flash，例如：其配置信息不需要存储在非易失性内存中，其配置信息已安全备份，或仅出于某些调试原因等。
Wi-Fi AMPDU

ESP32 同时支持接收和发送 AMPDU，AMPDU 可以大大提高 Wi-Fi 的吞吐量。
通常，应使能 AMPDU。禁用 AMPDU 通常用于调试目的。

4.33.33 故障排除

请见乐鑫 Wireshark 使用指南。

乐鑫 Wireshark 使用指南

1. 概述

1.1 什么是 Wireshark？ Wireshark（原称 Ethereal）是一个网络封包分析软件。网络封包分析软件的功能是撷取网络封包，并尽可能显示出最为详细的网络封包资料。Wireshark 使用 WinPCAP 作为接口，直接与网卡进行数据报文交换。
网络封包分析软件的功能可想像成“电工技师使用电表来量测电流、电压、电阻”的工作，只是将场景移植到网络上，并将电线替换成网线。
在过去，网络封包分析软件是非常昂贵，或是专门属于军利用的软件。Wireshark 的出现改变了这一切。
在 GNU GPL 通用许可证的保障范围内，使用者可以以免费的代价取得软件与其源代码，并拥有针对其源代码修改及客制化的权利。
Wireshark 是目前全世界最广泛的网络封包分析软件之一。

1.2 Wireshark 的主要应用 下面是 Wireshark 一些应用的举例：

- 网络管理员用来解决网络问题
- 网络安全工程师用来检测安全隐患
- 开发人员用来测试协议执行情况
- 用来学习网络协议

除了上面提到的，Wireshark 还可以用在其它许多场合。

1.3 Wireshark 的特性

- 支持 UNIX 和 Windows 平台
- 在接口实时捕捉包
- 能详细显示包的详细协议信息
- 可以打开/保存捕捉的包
- 可以导入导出其他捕捉程序支持的包数据格式
- 可以通过多种方式过滤包
- 多种方式查找包
- 通过过滤以多种色彩显示包
- 创建多种统计分析
- 等等
1.4 Wireshark 的“能”与“不能”？

- 捕获多种网络接口
  Wireshark 可以捕获多种网络接口类型的包，哪怕是无线局域网接口。
- 支持多种其它程序捕获的文件
  Wireshark 可以打开多种网络分析软件捕获的包。
- 支持多格式输出
  Wireshark 可以将捕获文件输出为多种其他捕获软件支持的格式。
- 对多种协议解码提供支持
  Wireshark 可以支持许多协议的解码。
- Wireshark 不是入侵检测系统
  如果您的网络中存在任何可疑活动，Wireshark 并不会主动发出警告。不过，当您希望对这些可疑活动一探究竟时，Wireshark 可以发挥作用。
- Wireshark 不会处理网络事务，它仅仅是“测量”（监视）网络
  Wireshark 不会发送网络包或做其它交互性的事情（名称解析除外，但您也可以禁止解析）。

2. 如何获取 Wireshark 官网链接：https://www.wireshark.org/download.html

Wireshark 支持多种操作系统，请在下载安装文件时，注意选择与您所用操作系统匹配的安装文件。

3. 使用步骤 本文档仅以 Linux 系统下的 Wireshark（版本号：2.2.6）为例。

1) 启动 Wireshark

Linux 下，可编辑一个 Shell 脚本，运行该文件即可开启 Wireshark 配置抓包网卡和信道。Shell 脚本如下：

```bash
ifconfig $1 down
iwconfig $1 mode monitor
iwconfig $1 channel $2
ifconfig $1 up
Wireshark
```

脚本中有两个参数：$1 和 $2，分别表示网卡和信道，例如，./xxx.sh wlan0 6 （此处，wlan0 即为抓包使用的网卡，后面的数字 6 即为 AP 或 soft-AP 所在的 channel）。

2) 运行 Shell 脚本开启 Wireshark，会出现 Wireshark 抓包开始界面

![Wireshark界面](image)

图 64: Wireshark 抓包界面

3) 选择接口，开始抓包

从上图红色框中可以看到有多个接口，第一个为本地网卡，第二个为无线网络。
可根据自己的需求选取相应的网卡，本文是以利用无线网卡抓取空中包为例进行简单说明。
双击 `wlan0` 即可开始抓包。

4) 设置过滤条件

抓包过程中会抓取到同信道所有的空包，但其实很多都是我们不需要的，因此很多时候我们会设置抓包的过滤条件从而得到我们想要的包。

下图中红色框内即为设置 `filter` 的位置。

![Wireshark Filter](image)

图 65: 设置 Wireshark 过滤条件

点击 `Filter` 按钮（下图的左上角蓝色按钮）会弹出 `display filter` 对话框。

![Display Filter](image)

图 66: Display Filter 对话框

点击 `Expression` 按钮，会出现 `Filter Expression` 对话框，在此你可以根据需求进行 `filter` 的设置。

最直接的方法：直接在工具栏上输入过滤条件。

点击在此区域输入或修改显示的过滤字符。在输入过程中会进行语法检查，如果输入的格式不正确，或者未输入完成，则背景显示为红色。直到您输入合法的表达式，背景变为绿色。你可以点击下拉列表选择您先前键入的过滤字符，列表会一直保留，即使你重新启动程序。

例如：下图所示，直接输入 2 个 MAC 作为过滤条件，点击 `Apply`（即图中的蓝色箭头），则表示只抓取 2 个此 MAC 地址之间的交互的包。

5) 封包列表

若想查看包的具体的信息只需要选中要查看的包，在界面的下方会显示出包的具体的格式和包的内容。如上图所示，我要查看第 1 个包，选中此包，图中红色框中即为包的具体内容。

6) 停止/开始包的捕捉

若要停止当前抓包，点击下图的红色按钮即可。
图 67: Filter Expression 对话框

图 68: 过滤条件工具栏

图 69: 在过滤条件工具栏中运用 MAC 地址过滤示例
图 70: 封包列表具体信息示例

图 71: 停止包的捕捉

图 72: 开始或继续包的捕捉
若要重新开始抓包，点击下图左上角的蓝色按钮即可。

7) 保存当前捕捉包

Linux下，可以通过依次点击“File”->“Export Packet Dissections”->“As Plain Text File”进行保存。

图73: 保存捕捉包

上图中，需要注意的是，选择All packets、Displayed以及All expanded三项。

Wireshark捕捉的包可以保存为其原生格式文件（libpcap），也可以保存为其他格式（如.txt文件）供其他工具进行读取分析。

4.34 Wi-Fi Security

4.34.1 ESP32 Wi-Fi Security Features

- Support for Protected Management Frames (PMF)
- Support for WPA3-Personal

In addition to traditional security methods (WEP/WPA-TKIP/WPA2-CCMP), ESP32 Wi-Fi supports state-of-the-art security protocols, namely Protected Management Frames based on 802.11w standard and Wi-Fi Protected Access 3 (WPA3-Personal). Together, PMF and WPA3 provide better privacy and robustness against known attacks on traditional modes.

4.34.2 Protected Management Frames (PMF)

Introduction

In Wi-Fi, management frames such as beacons, probes, (de)authentication, (dis)association are used by non-AP stations to scan and connect to an AP. Unlike data frames, these frames are sent unencrypted. An attacker can use eavesdropping and packet injection to send spoofed (de)authentication/(dis)association frames at the right time, leading to the following attacks in case of unprotected management frame exchanges.

- DOS attack on one or all clients in the range of the attacker.
- Tearing down existing association on AP side by sending association request.
- Forcing a client to perform 4-way handshake again in case PSK is compromised in order to get PTK.
- Getting SSID of hidden network from association request.
• Launching man-in-the-middle attack by forcing clients to deauth from legitimate AP and associating to a rogue one.

PMF provides protection against these attacks by encrypting unicast management frames and providing integrity checks for broadcast management frames. These include deauthentication, disassociation and robust management frames. It also provides Secure Association (SA) teardown mechanism to prevent spoofed association/authentication frames from disconnecting already connected clients.

There are 3 types of PMF configuration modes on both station and AP side -

• PMF Optional
• PMF Required
• PMF Disabled

Depending on PMF configurations on Station and AP side, the resulting connection will behave differently. The table below summarises all possible outcomes -

<table>
<thead>
<tr>
<th>STA Setting</th>
<th>AP Setting</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMF Optional</td>
<td>PMF Optional/Required</td>
<td>Mgmt Frames Protected</td>
</tr>
<tr>
<td>PMF Optional</td>
<td>PMF Disabled</td>
<td>Mgmt Frames Not Protected</td>
</tr>
<tr>
<td>PMF Required</td>
<td>PMF Optional/Required</td>
<td>Mgmt Frames Protected</td>
</tr>
<tr>
<td>PMF Required</td>
<td>PMF Disabled</td>
<td>STA refuses Connection</td>
</tr>
<tr>
<td>PMF Disabled</td>
<td>PMF Optional/Disabled</td>
<td>Mgmt Frames Not Protected</td>
</tr>
<tr>
<td>PMF Disabled</td>
<td>PMF Required</td>
<td>AP refuses Connection</td>
</tr>
</tbody>
</table>

API & Usage

ESP32 supports PMF in both Station and SoftAP mode. For both, the default mode is PMF Optional and disabling PMF is not possible. For even higher security, PMF required mode can be enabled by setting the required flag in pmf_cfg while using the esp_wifi_set_config() API. This will result in the device only connecting to a PMF enabled device and rejecting others.

注意: capable flag in pmf_cfg is deprecated and set to true internally. This is to take the additional security benefit of PMF whenever possible.

4.34.3 WPA3-Personal

Introduction

Wi-Fi Protected Access-3 (WPA3) is a set of enhancements to Wi-Fi access security intended to replace the current WPA2 standard. It includes new features and capabilities that offer significantly better protection against different types of attacks. It improves upon WPA2-Personal in following ways:

• WPA3 uses Simultaneous Authentication of Equals (SAE), which is password-authenticated key agreement method based on Diffie-Hellman key exchange. Unlike WPA2, the technology is resistant to offline-dictionary attack, where the attacker attempts to determine shared password based on captured 4-way handshake without any further network interaction.
• Disallows outdated protocols such as TKIP, which is susceptible to simple attacks like MIC key recovery attack.
• Mandates Protected Management Frames (PMF), which provides protection for unicast and multicast robust management frames which include Disassoc and Deauth frames. This means that the attacker cannot disrupt an established WPA3 session by sending forged Assoc frames to the AP or Deauth/Disassoc frames to the Station.
• Provides forward secrecy, which means the captured data cannot be decrypted even if password is compromised after data transmission.

Please refer to Security section of Wi-Fi Alliance’s official website for further details.
### Setting up WPA3 with ESP32

In IDF Menuconfig under Wi-Fi component, a config option “Enable WPA3-Personal” is provided to Enable/Disable WPA3. By default it is kept enabled, if disabled ESP32 will not be able to establish a WPA3 connection. Currently, WPA3 is supported only in the Station mode. Additionally, since PMF is mandated by WPA3 protocol, PMF Mode should be set to either Optional or Required while setting WiFi config.

Refer to *Protected Management Frames (PMF)* on how to set this mode.

After these settings are done, Station is ready to use WPA3-Personal. Application developers need not worry about the underlying security mode of the AP. WPA3-Personal is now the highest supported protocol in terms of security, so it will be automatically selected for the connection whenever available. For example, if an AP is configured to be in WPA3 Transition Mode, where it will advertise as both WPA2 and WPA3 capable, Station will choose WPA3 for the connection with above settings. Note that Wi-Fi stack size requirement will increase 3kB when WPA3 is used.

### 4.35 RF 共存

#### 4.35.1 概览

ESP32 只支持一路 RF, Bluetooth (BT 和 BLE) 和 Wi-Fi 共享这一路 RF，无法同时收发数据，因此采用时分复用的方法进行收发数据包。

#### 4.35.2 ESP32 支持的共存场景

<table>
<thead>
<tr>
<th>Wi-Fi</th>
<th>STA</th>
<th>BLE</th>
<th>Scan</th>
<th>Advertising</th>
<th>Connecting</th>
<th>Connected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connecting</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connected</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>SOFTAP</td>
<td>TX Beacon</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connecting</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connected</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>Sniffer</td>
<td>RX</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>ESP-NOW</td>
<td>RX</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TX</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wi-Fi</th>
<th>STA</th>
<th>BR/EDR</th>
<th>Inquiry</th>
<th>Inquiry scan</th>
<th>Page</th>
<th>Page scan</th>
<th>Connected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Scan</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connecting</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connected</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>SOFTAP</td>
<td>TX Beacon</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connecting</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connected</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>Sniffer</td>
<td>RX</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>ESP-NOW</td>
<td>RX</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TX</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
4.35.3 共存机制与策略

### 共存机制

基于优先级抢占的 RF 资源分配机制，如下图所示。Bluetooth 模块和 Wi-Fi 模块向共存模块申请 RF 资源，共存模块根据二者的优先级高低裁决 RF 归谁使用。

![共存机制示意图](image)

图 74: 共存机制

### 共存策略

**共存周期和时间片**  Wi-Fi、BT、BLE 三者对于 RF 的使用，主要是按照时间片来划分的。在一个共存周期内，按照 Wi-Fi、BT、BLE 的顺序划分时间片。在 Wi-Fi 的时间片内，Wi-Fi 会向共存仲裁模块发出较高优先级的请求，同理，BT/BLE 在自己的时间片内会具有较高优先级。共存周期大小和各个时间片占比根据 Wi-Fi 的状态分成四类：

1. **IDLE** 状态：BT 和 BLE 共存由 Bluetooth 模块控制。
2. **CONNECTED** 状态：共存周期以目标信标传输时间 (Target Beacon Transmission Time, TBTT) 为起点，周期大于 100 ms。
3. **SCAN** 状态：Wi-Fi 时间片以及共存周期都比在 CONNECTED 状态下的长，为了确保蓝牙的性能，蓝牙的时间片也会做相应的调整。
4. **CONNECTING** 状态：Wi-Fi 时间片在 CONNECTED 状态下的长。为了确保蓝牙的性能，蓝牙的时间片也会做相应的调整。

共存逻辑会根据当前 Wi-Fi 和 Bluetooth 的使用场景来选取不同的共存周期和共存时间片的划分策略。对应一个使用场景的共存策略，我们称之为 “共存模板”。比如，Wi-Fi CONNECTED 与 BLE CONNECTED 的场景，就对应有一个共存模板。在这个共存模板中，一个共存周期内 Wi-Fi 和 BLE 的时间片各占 50%，时间分配如下图所示：

**动态优先级** 共存模块对 Wi-Fi 和 Bluetooth 不同的状态赋予其不同的优先级。每种状态下的优先级并不是一成不变的，例如每次广播事件 (Advertising event) 中会有一个广播事件使用高优先级。如果高优先级的广播事件发生在 Wi-Fi 时间片内，RF 的使用权可能会被 BLE 抢占。

备注：Y：支持且性能稳定。C1：不能保证性能处于稳定状态。X：不支持。S：在 STA 模式下支持且性能稳定，否则不支持。
Wi-Fi 非连接模块的共存 在一定程度上，某些 Wi-Fi 非连接模块功耗参数 Window 与 Interval 的组合会导致共存模块在 Wi-Fi 时间片外申请共存优先级。这是为了使设定的功耗参数在共存时获取 RF 资源。但这也会影响剩余的蓝牙性能。

如果 Wi-Fi 非连接模块功耗参数为默认值时，上述动作不会发生。共存模块会按照性能稳定的模式运行。因此，除非您对特定非连接功耗参数下的共存性能有足够的测试，请在共存场景下将 Wi-Fi 非连接模块功耗参数配置为默认参数。

请参考 非连接模块功耗管理 获取更多信息。

### 4.35.4 如何使用共存功能

**共存 API 的使用**

在大多数共存情况下，ESP32 会自动进行共存状态切换，无需调用 API 对其进行干预。但是对于 BLE MESH 和 Wi-Fi 的共存，ESP32 对其提供了两个 API。当 BLE MESH 的状态发生变化时，应先调用 esp_coex_status_bit_clear 对上一个状态进行清除，然后调用 esp_coex_status_bit_set 设置当前状态。

**BLE MESH 共存状态描述** 由于 Wi-Fi 和 Bluetooth 固件无法获取当前的上层应用的场景，一些共存模块需要应用代码调用共存的 API 才能生效。BLE MESH 工作状态就需要由应用层通知给共存模块，用于选择共存模块。

- ESP_COEX_BLE_ST_MESH_CONFIG：正在组网。
- ESP_COEX_BLE_ST_MESH_TRAFFIC：正在传输数据。
- ESP_COEX_BLE_ST_MESH_STANDBY：处于空闲状态，无大量数据交互。

**共存 API 错误代码**

所有共存 API 都有自定义的返回值，即错误代码。这些代码可分类为：

- 无错误，例如：返回值为 ESP_OK 代表 API 成功返回。
- 可恢复错误，例如：返回值为 ESP_ERR_INVALID_ARG 代表 API 参数错误。

**设置共存编译时选项**

- 在完成共存程序编写的时候，您必须通过 menuconfig 选择 CONFIG_ESP32_WIFI_SW_COEXIST_ENABLE 打开软件共存配置选项，否则就无法使用上文提到的共存功能。
- 为了在共存情况下获得更好的 Wi-Fi 和蓝牙的通信性能，建议将 Wi-Fi 协议栈的 task 和蓝牙 Controller 以及 Host 协议栈的 task 运行在不同的 CPU 上。您可以通过 CONFIG_BT_DM_CTRL_PINNED_TO_CORE_CHOICE 和 CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE（或者 CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE）选择将蓝牙 controller 以及 host 协议栈的 task 放在同一个 CPU 上，再通过 CONFIG_ESP32_WIFI_TASK_CORE_ID 选择将 Wi-Fi 协议栈 task 放在另一个 CPU 上。
Chapter 4. API 指南

• 在 共 存 情 况 下 BLE SCAN 可 能 会 被 Wi-Fi 打 断 且 Wi-Fi 在 当 前 的 BLE scan window 结 束
前 释 放 了 RF 资 源。 为 了 使 BLE 在 当 前 的 scan window 内 再 次 获 取 RF 资 源， 您 可 以 通
过CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED 选择打开 FULL SCAN 配置选项。
• 您可以通过修改以下 menuconfig 选项，以减小内存开销：
1) CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY 选择打开蓝牙协议栈动态内存配置选项。
2) CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM 选择减少 Wi-Fi 静态接收数据缓冲区的数量。
3) CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM 选择减少 Wi-Fi 动态接收数据缓冲区的数
量。
4) CONFIG_ESP32_WIFI_TX_BUFFER 选择使用动态分配发送数据缓冲区配置选项。
5) CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM 选择减少 Wi-Fi 动态发送数据缓冲区的数
量。
6) CONFIG_ESP32_WIFI_TX_BA_WIN 选择减少 Wi-Fi Block Ack TX 窗口的数量。
7) CONFIG_ESP32_WIFI_RX_BA_WIN 选择减少 Wi-Fi Block Ack RX 窗口的数量。
8) CONFIG_ESP32_WIFI_MGMT_SBUF_NUM 选择减少 Wi-Fi 管理短缓冲区的数量。
9) CONFIG_ESP32_WIFI_RX_IRAM_OPT 选择关闭此配置选项，关闭此配置选项将会减少大约 17
KB 的 IRAM 内存。
10) CONFIG_LWIP_TCP_SND_BUF_DEFAULT 选择减小 TCP 套接字默认发送缓存区大小。
11) CONFIG_LWIP_TCP_WND_DEFAULT 选择减小 TCP 套接字默认接收窗口的大小。
12) CONFIG_LWIP_TCP_RECVMBOX_SIZE 选择减小 TCP 接收邮箱的大小。
13) CONFIG_LWIP_UDP_RECVMBOX_SIZE 选择减小 UDP 接收邮箱的大小。
14) CONFIG_LWIP_TCPIP_RECVMBOX_SIZE 选择减小 TCPIP 任务接收邮箱大小。
备注: 由于共存配置选项依赖于蓝牙配置选项，所以请先打开蓝牙配置选项，然后在 Wi-Fi 配置选项中
打开共存配置选项。

4.36

Reproducible Builds

4.36.1

Introduction

ESP-IDF build system has support for reproducible builds.
When reproducible builds are enabled, the application built with ESP-IDF doesn’t depend on the build environment.
Both the .elf file and .bin files of the application remains exactly the same, even if the following variables change:
• Directory where the project is located
• Directory where ESP-IDF is located (IDF_PATH)
• Build time

4.36.2

Reasons for non-reproducible builds

There are several reasons why an application may depend on the build environment, even when the same source code
and tools versions are used.
• In C code, __FILE__ preprocessor macro is expanded to the full path of the source file.
• __DATE__ and __TIME__ preprocessor macros are expanded to compilation date and time.
• When the compiler generates object files, it adds sections with debug information. These sections help debuggers, like GDB, to locate the source code which corresponds to a particular location in the machine code.
These sections typically contain paths of relevant source files. These paths may be absolute, and will include
the path to ESP-IDF or to the project.
There are also other possible reasons, such as unstable order of inputs and non-determinism in the build system.

4.36.3

Enabling reproducible builds in ESP-IDF

Reproducible builds can be enabled in ESP-IDF using CONFIG_APP_REPRODUCIBLE_BUILD option.
Espressif Systems

2447
Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151


This option is disabled by default. It can be enabled in menuconfig.
The option may also be added into `sdkconfig.defaults`. If adding the option into `sdkconfig.defaults`, delete the `sdkconfig` file and run the build again. See self-defined `sdkconfig` defaults for more information.

### 4.36.4 How reproducible builds are achieved

ESP-IDF achieves reproducible builds using the following measures:

- In ESP-IDF source code, `__DATE__` and `__TIME__` macros are not used when reproducible builds are enabled. Note, if the application source code uses these macros, the build will not be reproducible.
- ESP-IDF build system passes a set of `-fmacro-prefix-map` and `-fdebug-prefix-map` flags to replace base paths with placeholders:
  - Path to ESP-IDF is replaced with `/IDF`
  - Path to the project is replaced with `/IDF_PROJECT`
  - Path to the build directory is replaced with `/IDF_BUILD`
  - Paths to components are replaced with `/COMPONENT_NAME_DIR` (where `NAME` is the name of the component)
- Build date and time are not included into the application metadata structure if `CONFIG_APP_REPRODUCIBLE_BUILD` is enabled.
- ESP-IDF build system ensures that source file lists, component lists and other sequences are sorted before passing them to CMake. Various other parts of the build system, such as the linker script generator also perform sorting to ensure that same output is produced regardless of the environment.

### 4.36.5 Reproducible builds and debugging

When reproducible builds are enabled, file names included in debug information sections are altered as shown in the previous section. Due to this fact, the debugger (GDB) is not able to locate the source files for the given code location.

This issue can be solved using GDB `set substitute-path` command. For example, by adding the following command to GDB init script, the altered paths can be reverted to the original ones:

```plaintext
set substitute-path /COMPONENT_FREERTOS_DIR /home/user/esp/esp-idf/components/
```

ESP-IDF build system generates a file with the list of such `set substitute-path` commands automatically during the build process. The file is called `prefix_map_gdbinit` and is located in the project build directory.

When `idf.py gdb` is used to start debugging, this additional gdbinit file is automatically passed to GDB. When launching GDB manually or from and IDE, please pass this additional gdbinit script to GDB using `-x build/prefix_map_gdbinit` argument.

### 4.36.6 Factors which still affect reproducible builds

Note that the built application still depends on:

- ESP-IDF version
- Versions of the build tools (CMake, Ninja) and the cross-compiler

`IDF Docker Image` can be used to ensure that these factors do not affect the build.

### 4.37 C++ Support

ESP-IDF is primarily written in C and provides C APIs. However, ESP-IDF supports development of applications in C++. This document covers various topics relevant to C++ development.

The following C++ features are supported:
Chapter 4. API

- Exception handling
- C++ language standard
- Runtime Type Information (RTTI)
- Thread Local Storage (thread_local keyword)
- All C++ features implemented by GCC, except for some limitations. See GCC documentation for details on features implemented by GCC.

4.37.1 esp-idf-cxx Component

esp-idf-cxx component provides higher-level C++ APIs for some of the ESP-IDF features. This component is available from the IDF Component Registry.

4.37.2 C++ language standard

By default, ESP-IDF compiles C++ code with C++20 language standard with GNU extensions (-std=gnu++20).

To compile the source code of a certain component using a different language standard, set the desired compiler flag in the component CMakeLists.txt file:

```cpp
idf_component_register(...) target_compile_options(${COMPONENT_LIB} PRIVATE -std=gnu++2b)
```

Use PUBLIC instead of PRIVATE if the public header files of the component also need to be compiled with the same language standard.

4.37.3 Multithreading

C++ threads, mutexes, and condition variables are supported. C++ threads are built on top of pthreads, which in turn wrap FreeRTOS tasks.

See cxx/pthread for an example of creating threads in C++.

4.37.4 Exception handling

Support for C++ Exceptions in ESP-IDF is disabled by default, but can be enabled using the CONFIG_COMPILER_CXX.Exceptions option.

If an exception is thrown, but there is no catch block, the program will be terminated by the abort function, and the backtrace will be printed. See Fatal Errors for more information about backtraces.

C++ Exceptions should only be used for exceptional cases, something happening unexpectedly and that is quite rare, e.g. an event that happens less frequently than 1 every 100 times. Do not use them for control flow (see also the section about resource usage below)! For more information on how to use C++ Exceptions, see the ISO C++ FAQ and CPP Core Guidelines.

See cxx/exceptions for an example of C++ exception handling.

### C++ Exception Handling and Resource Usage

Enabling exception handling normally increases application binary size by a few KB.

Additionally, it may be necessary to reserve some amount of RAM for exception emergency pool. Memory from this pool will be used if it is not possible to allocate exception object from the heap. The amount of memory in the emergency pool can be set using the CONFIG_COMPILER_CXX.Exceptions.EMG_POOL_SIZE variable. Some additional stack memory (around 200 bytes) will also be used if and only if a C++ Exception is actually thrown, because it requires calling some functions from the top of the stack to initiate exception handling.
The run time of code using C++ exceptions depends on what actually happens at run time. If no exception is thrown, the code tends to be somewhat faster since there is no need to check error codes. If an exception is thrown, the run time of the code that handles exceptions will be orders of magnitude slower than code returning an error code. This increase may or may not be significant, however, in the entire application, in particular if the error handling requires additional action, such as a user input or messaging to a cloud. But exception-throwing code should never be used in real-time critical code paths.

### 4.37.5 Runtime Type Information (RTTI)

Support for RTTI is disabled by default, but can be enabled using `CONFIG_COMPILER_CXX_RTTI` option. Enabling this option compiles all C++ files with RTTI support enabled, which allows using `dynamic_cast` conversion and `typeid` operator. Enabling this option typically increases the binary size by tens of kB.

See `cxx/rtti` for an example of using RTTI in ESP-IDF.

### 4.37.6 Developing in C++

The following sections provide tips on developing ESP-IDF applications in C++.

**Combining C and C++ code**

When part of the application is developed in C and part in C++, it is important to understand the concept of language linkage.

In order for a C++ function to be callable from C code, it has to be both declared and defined with C linkage (`extern "C"`):

```c
// declaration in the header file:
#ifdef __cplusplus
extern "C" {    
#endif
void my_cpp_func(void);
#ifdef __cplusplus}
#endif
// definition in a .cpp file:
extern "C" void my_cpp_func(void) {
  // ...
}
```

In order for a C function to be callable from C++, it has to be declared with C linkage:

```c
// declaration in the header file:
#ifdef __cplusplus
extern "C" {    
#endif
void my_c_func(void);
#ifdef __cplusplus}
#endif
// definition in a .c file:
void my_c_func(void) {
```

(下页继续)
// ...

```cpp
extern "C" void app_main()
{
}
```

### Defining `app_main` in C++

ESP-IDF expects the application entry point, `app_main`, to be defined with C linkage. When `app_main` is defined in a .cpp source file, it has to be designated as `extern "C"`:

```cpp
extern "C" void app_main()
{
}
```

### Designated initializers

Many of the ESP-IDF components use `configuration structures` as arguments to the initialization functions. ESP-IDF examples written in C routinely use designated initializers to fill these structures in a readable and maintainable way.

C and C++ languages have different rules with regards to the designated initializers. For example, C++ language version C++20, currently the default in ESP-IDF, does not support out-of-order designated initialization, nested designated initialization, mixing of designated initializers and regular initializers, and designated initialization of arrays. Therefore, when porting ESP-IDF C examples to C++, some changes to the structure initializers may be necessary. See the C++ aggregate initialization reference for more details.

### `iostream`

`iostream` functionality is supported in ESP-IDF, with a couple of caveats:

1. Normally ESP-IDF build process eliminates the unused code. However in the case of `iostreams`, simply including `<iostream>` header in one of the source files significantly increases the binary size (by about 200 kB).
2. By default, ESP-IDF uses a simple non-blocking implementation of the standard input stream (stdin). To get the usual behavior of `std::cin`, the application has to initialize the UART driver and enable the blocking mode as shown in `common_components/protocol_examples_common/stdin_out.c`.

### 4.37.7 Limitations

- Linker script generator doesn’t support function level placements for functions with C++ linkage.
- Various section attributes (such as `IRAM_ATTR`) are ignored when used with template functions.
- Vtables are placed into Flash and are not accessible when the flash cache is disabled. Therefore, virtual function calls should be avoided in `IRAM-safe interrupt handlers`. Placement of Vtables cannot be adjusted using the linker script generator, yet.
- `C++ filesystem` features are not supported.

### 4.37.8 What to Avoid

Do not use `setjmp/longjmp` in C++! `longjmp` blindly jumps up the stack without calling any destructors, easily introducing undefined behavior and memory leaks. Use C++ exceptions instead, they will guarantee correctly calling destructors. If you cannot use C++ exceptions, use alternatives (except `setjmp/longjmp` themselves) such as simple return codes.
Chapter 5

迁移指南

5.1 迁移到 ESP-IDF 5.x

5.1.1 从 4.4 迁移到 5.0

经典蓝牙

Bluedroid

- bt/host/bluedroid/api/include/api/esp_hf_defs.h
  - 在 esp_hf_cme_err_t 中
    * ESP_HF_CME_MEMORY_FULL 改名为 ESP_HF_CME_MEMORY_FULL
    * ESP_HF_CME_MEMORY_FAILURE 改名为 ESP_HF_CME_MEMORY_FAILURE
- bt/host/bluedroid/api/include/api/esp_hf_api.h
  - esp_bt_hf_init(esp_bd_addr_t remote_addr) 改为 esp_hf_ag_init(void)
  - esp_bt_hf_deinit(esp_bd_addr_t remote_addr) 改为 esp_hf_ag_deinit(void)
  - esp_bt_hf_register_callback 改为 esp_hf_ag_register_callback
  - esp_bt_hf_connect 改为 esp_hf_ag_slc_connect
  - esp_bt_hf_disconnect 改为 esp_hf_ag_slc_disconnect
  - esp_bt_hf_connect_audio 改为 esp_hf_ag_audio_connect
  - esp_bt_hf_disconnect_audio 改为 esp_hf_ag_audio_disconnect
  - esp_bt_hf_vra 改为 esp_hf_ag_vra_control
  - esp_bt_hf_volume_control 改为 esp_hf_ag_volume_control
  - esp_hf_unat_response 改为 esp_hf_ag_unknown_at_send
  - esp_hf_cme_response 改为 esp_hf_ag_cme_send
  - esp_bt_hf_indchange_notification 改为 esp_hf_ag_devices_status_indchange
  - esp_bt_hf_cind_response 改为 esp_hf_ag_cind_response
  - esp_bt_hf_cops_response 改为 esp_hf_ag_cops_response
  - esp_bt_hf_clcc_response 改为 esp_hf_ag_clcc_response
  - esp_bt_hf_cnum_response 改为 esp_hf_ag_cnum_response
  - esp_bt_hf_bsir 改为 esp_hf_ag_bsir
  - esp_bt_hf_answer_call 改为 esp_hf_ag_answer_call
低功耗蓝牙

Bluedroid

以下 Bluedroid 宏、类型和函数已被重命名：

- bt/host/bluedroid/api/include/api/esp_gap_ble_api.h
  - esp_gap_ble_cb_event_t 中:
    - ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT 改名 为 ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT
    - ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT 改名 为 ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT
    - ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT 改名 为 ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT
- esp_ble_wpoperation_t 改名为 esp_ble_wpoperation_t
- esp_ble_gap_cb_param_t.pkt_data_lenth_cmpl 改名 为 pkt_data_length_cmpl
- esp_ble_gap_cb_param_t.update_whitelist_cmpl.wl_operation 改名 为 wl_operation
- esp_ble_gap_set_preferred_default_phy 改名为 esp_ble_gap_set_preferred_default_phy()

Nimble

以下 Nimble API 已被移除：

- bt/host/nimble-esp-hci/include/esp_nimble_hci.h
  - 移除 esp_err_t esp_nimble_hci_and_controller_init(void)
    - 控制器初始化，使能以及 HCI 初始化的调用已经被移至 nimble_port_init 中。可直接删除该函数。
  - 移除 esp_err_t esp_nimble_hci_and_controller_deinit(void)
    - 控制器去初始化，禁用以及 HCI 去初始化的调用已经被移至 nimble_port_deinit 中。可直接删除该函数。

ESP-BLE-MESH

以下 ESP-BLE-MESH 宏已被重命名：

- bt/esp_ble_mesh/api/esp_ble_mesh_defs.h
  - esp_ble_mesh_prov_cb_event_t 中:
    - ESP_BLE_MESH_PROVISIONER_DRIECT_ERASE_SETTINGS_COMP_EVT 改名为 ESP_BLE_MESH_PROVISIONER_DRIECT_ERASE_SETTINGS_COMP_EVT
迁移构建系统至 ESP-IDF v5.0

从 GNU Make 构建系统迁移至 ESP-IDF v5.0 ESP-IDF v5.0 已不再支持基于 Make 的工程，请参考从 ESP-IDF GNU Make 构建系统迁移至 CMake 构建系统 进行迁移。

更新片段文件语法 请参考 将链接器脚本片段文件语法迁移至 ESP-IDF v5.0 适用版本 对 v3.x 的语法进行更新。

明确指定组件依赖 在之前的 ESP-IDF 版本中，除了通用组件依赖外，还有一些组件总是作为公共依赖项在构建中被添加至每个组件中，如：

- driver
- efuse
- esp_timer
- lwip
- vfs
- esp_wifi
- esp_event
- esp_netif
- esp_eth
- esp_phy

这意味着可以直接包含这些组件的头文件，而无需在 idf_component_register 中将它们指定为依赖。此行为是由各种常见组件的传递依赖关系引起的。

在 ESP-IDF v5.0 中，此行为已修复，这些组件不再默认作为公共依赖项添加。

如果组件所依赖的某个组件不属于通用组件依赖项，则必须显式地声明此依赖关系。可以通过在组件的 CMakeLists.txt 中的 idf_component_register 调用中添加 REQUIRES <component_name> 或 PRIV_REQUIRES <component_name> 来完成。有关指定组件依赖的更多信息，请参阅组件依赖。

设置 COMPONENT_DIRS 和 EXTRA_COMPONENT_DIRS 变量 为了实现构建项目时的路径能够包含空格，ESP-IDF v5.0 做了一系列改进，其中包括改进了 CMakeLists.txt 文件中的 COMPONENT_DIRS 和 EXTRA_COMPONENT_DIRS 变量。

ESP-IDF v5.0 版本中，不再支持添加不存在的目录到变量 COMPONENT_DIRS 或 EXTRA_COMPONENT_DIRS 中，否则会出错。

同时，ESP-IDF v5.0 中也不再支持使用字符串拼接的方式定义 COMPONENT_DIRS 或 EXTRA_COMPONENT_DIRS 变量。这些变量应该定义为 CMake 列表。例如：

```cmake
set(EXTRA_COMPONENT_DIRS path1 path2)
lst(APPEND EXTRA_COMPONENT_DIRS path3)
```

不支持：

```cmake
set(EXTRA_COMPONENT_DIRS "path1 path2")
set(EXTRA_COMPONENT_DIRS "$\{EXTRA_COMPONENT_DIRS\} path3")
```

将这些变量定义为 CMake 列表的方式兼容之前的 ESP-IDF 版本。

更新 target_link_libraries 用法 ESP-IDF v5.0 修复了组件的 CMake 变量传播问题。此问题导致本应该只应用于某一组件的编译器标志和定义应用到了项目中的每个组件。

该修复也带来一定的副作用，从 ESP-IDF v5.0 开始，用户项目在使用 target_link_libraries 时必须明确指定 project_elf，同时自定义 CMake 项目必须指定 PRIVATE, PUBLIC 或 INTERFACE 参数。这是一项重大变更，不兼容以前的 ESP-IDF 版本。

例如：
更新 CMake 版本 在 ESP-IDF v5.0 中，最低 CMake 版本已更新到 3.16，并且不再支持低于 3.16 的版本。如果您的操作系统没有安装 CMake，请运行 tools/idf_tools.py install cmake 来安装合适的版本。

该变更会影响到使用系统提供的 CMake 以及自定义 CMake 的 ESP-IDF 用户。

重新定义特定目标配置文件的应用顺序 ESP-IDF v5.0 重新安排了特定目标配置文件和 SDKCONFIG_DEFAULTS 中所有其他文件的应用顺序。现在，特定目标的配置文件将在引入它的文件之后，在 SDKCONFIG_DEFAULTS 中后续的其他文件之前应用。

例如:

```bash
如果 `''SDKCONFIG_DEFAULTS="sdkconfig.defaults;sdkconfig_devkit1
"'`，且同一文件夹内有 `''sdkconfig.defaults.esp32`` 文件，那么文件的应用顺序为: (1) sdkconfig.defaults (2) sdkconfig.defaults.esp32 (3) sdkconfig_devkit1
```

如果某个键在不同的特定目标配置文件中有不同的值，那么后者的值会覆盖前者。例如在以上案例中，如果某个键在 sdkconfig.defaults.esp32 和 sdkconfig_devkit1 中的值不同，则在 sdkconfig_devkit1 中的值会覆盖在 sdkconfig.defaults.esp32 中的值。

如果确实需要设置特定目标的配置值，请将其放到后应用的特定目标文件中，如 sdkconfig_devkit1.esp32。

GCC

GCC 版本 ESP-IDF 之前使用的 GCC 版本为 8.4.0，现已针对所有芯片目标升级至 GCC 11.2.0。若需要将您的代码从 GCC 8.4.0 迁移到 GCC 11.2.0，请参考以下官方 GCC 迁移指南。

- 迁移至 GCC 9
- 迁移至 GCC 10
- 迁移至 GCC 11

警告 升级至 GCC 11.2.0 后会触发新警告，或是导致原有警告内容发生变化。所有 GCC 警告的详细内容，请参考 GCC 警告选项。建议用户仔细检查代码，并设法解决这些警告。但由某些警告的特殊性及用户代码的复杂性，有些警告可能为误报，需要进行关键修复。在这种情况下，用户可以采取多种方式来抑制这些警告。本节介绍了用户可能遇到的常见警告及如何抑制这些警告。

注意：建议用户在抑制警告之前仔细确认该警告是否确实为误报。

-Wstringop-overflow、-Wstringop-overread、-Wstringop-truncation 和 -Warray-bounds 如果编译器不能准确判断内存或字符串的大小，使用 memory/string copy/compare 函数的用户会遇到某种 -Wstringop 警告。下文展示了触发这些警告的代码，并介绍了如何抑制这些警告。
Chapter 5. 迁移指南

```c
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wstringop-overflow"
#pragma GCC diagnostic ignored "-Warray-bounds"
memset(RTC_SLOW_MEM, 0, CONFIG_ULP_COPROC_RESERVE_MEM); // <<-- 此行触发了警告
#pragma GCC diagnostic pop
```

```c
#pragma GCC diagnostic push
#if __GNUC__ >= 11
#pragma GCC diagnostic ignored "-Wstringop-overread" // <<-- 此键从 GCC 11 开始引入
#endif
#pragma GCC diagnostic ignored "-Warray-bounds"
memcpy(backup_write_data, (void*)EFUSE_PGM_DATA0_REG, sizeof(backup_write_data)); // <<-- 此行触发了警告
#pragma GCC diagnostic pop
```

-Waddress-of-packed-member 当访问打包 struct 中的某个未对齐成员时，由于非对齐内存访问会对性能产生影响，GCC 会触发-Waddress-of-packed-member 警告。然而，所有基于 Xtensa 或 RISC-V 架构的 ESP 芯片都允许非对齐内存访问，并且不会产生额外的性能影响。因此，在大多数情况下，可以忽略此问题。

```c
components/bt/host/bluedroid/btc/profile/std/gatt/btc_gatt_util.c: In function btc_to_bta_gatt_id:
components/bt/host/bluedroid/btc/profile/std/gatt/btc_gatt_util.c:105:21: warning:
   taking address of packed member of 'struct <anonymous>' may result in an
   unaligned pointer value [-Waddress-of-packed-member]
105 | btc_to_bta_uuid(&p_dest->uuid, &p_src->uuid);
    | ^~~~~~~~~~~~~
```

如果该警告在多个源文件中多次出现，可以在 CMake 级别抑制该警告，如下所示。

```cmake
set_source_files_properties(
    "host/bluedroid/bta/gatt/bta_gatcc_act.c"
    "host/bluedroid/bta/gatt/bta_gatcc_cache.c"
    "host/bluedroid/btc/profile/std/gatt/btc_gatt_util.c"
    "host/bluedroid/btc/profile/std/gatt/btc_gatts.c"
    PROPERTIES COMPILE_FLAGS "-Wno-address-of-packed-member"
)
```

但如果只有一或两处警告，可以直接在源代码中进行抑制，如下所示。

```c
#pragma GCC diagnostic push
#if __GNUC__ >= 9
#pragma GCC diagnostic ignored "-Waddress-of-packed-member" // <<-- 此键从 GCC 11 开始引入
#endif
#pragma GCC diagnostic ignored "-Warray-bounds"
uint32_t* reg_ptr = (uint32_t*)src;
#pragma GCC diagnostic pop
```

llabs() 用于 64 位整数 stdlib.h 中的函数 abs() 需要使用 int 参数。请在计划为 64 位的类型中使用 llabs()，尤其是 time_t。乐鑫工具链更新

Xtensa 编译器中的 int32_t 和 uint32_t 在 Xtensa 编译器中，int32_t 和 uint32_t 类型已分别从 int 和 unsigned int 更新为 long 和 unsigned long。此更新应与上游 GCC 相匹配，上游 GCC 在 Xtensa、RISC-V 和其他架构上使用 long 整数来表示 int32_t 和 uint32_t。
<table>
<thead>
<tr>
<th>Xtensa</th>
<th>2021r2 及以上版本，GCC 8</th>
<th>2022r1，GCC 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>riscv32</td>
<td>(unsigned) int</td>
<td>(unsigned) long</td>
</tr>
<tr>
<td></td>
<td>(unsigned) long</td>
<td>(unsigned) long</td>
</tr>
</tbody>
</table>

上述变化主要影响到使用 `<inttypes.h>` 提供的类型来格式化字符串的代码。请使用 PRIi32、PRIxx 等占位符来分别替换 %i、%x 等。

在其他情况下，请注意枚举支持 int 类型。

通常，int32_t 和 int 为不同的类型。同样，uint32_t 和 unsigned int 也为不同的类型。

如果用户在其应用中对格式化字符串进行上述更新，程序会报错，如下所示：

```c
#include/esp_diagnostics.h:238:29: error: format '%u' expects argument of type 'unsigned int', but argument 3 has type 'uint32_t' (aka 'long unsigned int') [-Werror=format=
```

移除构建选项 CONFIG_COMPILER_DISABLE_GCC8_WARNINGS 原有的 CONFIG_COMPILER_DISABLE_GCC8_WARNINGS 选项用于构建使用现已弃用的 GCC 5 工具链编写的陈旧代码。但由于已经过去较长时间，现在可以对警告进行修复，因此该选项已被移除。

目前，在 GCC 11 中，建议用户仔细检查代码，尽量解决编译器警告。

### 网络

**esp_eth_ioctl() API** 此前，`esp_eth_ioctl()` API 存在以下问题：

- 在某些情况下，第三个参数 (数据类型为 void /*) 可以接受 int/bool 类型实参 (而非指针) 作为输入。然而，文档中未描述这些情况。
- 为了将 int/bool 类型实参作为第三个参数传递，实参将被强制转换为 void * 类型，以防出现如下所示的编译器警告。此等转换可能引发 `esp_eth_ioctl()` 函数的滥用。

```c
esp_eth_ioctl(eth_handle, ETH_CMD_S_FLOW_CTRL, (void *)true);
```

因此，我们统一了 `esp_eth_ioctl()` 的用法。现在，该结构体的第三个参数在传递时必须作为指向特定数据类型的指针，表示 `esp_eth_ioctl()` 调用/存储数据的位置。`esp_eth_ioctl()` 的用法如下列代码所示。

设置以太网配置的用例如下：

```c
eth_duplex_t new_duplex_mode = ETH_DUPLEX_HALF;
esp_eth_ioctl(eth_handle, ETH_CMD_S_DUPLEX_MODE, &new_duplex_mode);
```

获取以太网配置的用例如下：

```c
void esp_eth_ioctl(eth_handle, ETH_CMD_S_DUPLEX_MODE, &new_duplex_mode);
```
KSZ8041/81 和 LAN8720 驱动更新

KSZ8041/81 和 LAN8720 驱动现已更新，以支持相关产品系列中的更多设备（如新一代设备）。上述驱动能够识别特定芯片编号及驱动提供的潜在支持。

更新之后，通用函数将替代特定“芯片编号”函数得以调用：
- 删除 esp_eth_phy_new_ksz8041() 以及 esp_eth_phy_new_ksz8081()，转而使用esp_eth_phy_new_ksz80xx()
- 删除 esp_eth_phy_new_lan8720()，转而使用esp_eth_phy_new_lan87xx()

ESP NETIF Glue 事件处理程序 esp_eth_set_default_handlers() 和 esp_eth_clear_default_handlers() 函数现已删除。现在可以自动处理以太网默认 IP 层处理程序的注册。如您在注册以太网/IP 事件处理程序之前已经按照建议完全初始化以太网驱动和网络接口，则无需执行任何操作（除了删除受影响的函数）。否则，在注册用户事件处理程序后，应随即启动以太网驱动。

PHY 地址自动检测 以太网 PHY 地址自动检测函数 esp_eth_detect_phy_addr() 已重命名为esp_eth_phy_802_3_detect_phy_addr()，其声明移至 esp_eth/include/esp_eth_phy_802_3.h。

SPI 以太网模块初始化 以太网模块的初始化过程已经简化。此前，您需要在实例化 SPI 以太网 MAC 之前，使用spi_bus_add_device() 手动分配 SPI 设备。

现在，由于 SPI 设备已在内部分配，您无需再调用spi_bus_add_device()。eth_dm9051_config_t, eth_w5500_config_t 和 eth_ksz8851snl_config_t 配置结构体现已包含 SPI 设备配置成员（例如，可以微调可能依赖 PCB 设计的 SPI 时序）。ETH_DM9051_DEFAULT_CONFIG, ETH_W5500_DEFAULT_CONFIG 和 ETH_KSZ8851SNL_DEFAULT_CONFIG 配置初始化宏也已接受新的参数输入。了解 SPI 以太网模块初始化示例，请查看以太网 API 参考指南。

TCP/IP 适配器 TCP/IP 适配器是在 ESP-IDF v4.1 之前使用的网络接口抽象组件。本文档概述了从 tcpip_adapter API 移迁至ESP-NETIF 的过程。

更新网络连接代码

网络软件栈初始化
- 您只需用 esp_netif_init() 替换 tcpip_adapter_init()，注意 esp_netif_init() 函数现将返回标准错误代码。了解详细信息，请参考ESP-NETIF。
- esp_netif_deinit() 函数用于反初始化网络软件栈。
- 您还需用 #include "esp_netif.h" 替换 #include "tcpip_adapter.h"。

创建网络接口 更新之前，TCP/IP 适配器静态定义了以下三个接口：
- Wi-Fi Station
- Wi-Fi AP
- 以太网

接口定义现已更新，网络接口的设计应严格参考ESP-NETIF，使其能够连接至 TCP/IP 软件栈。例如，在 TCP/IP 软件栈和事件循环初始化完成后，Wi-Fi 的初始化代码必须显示调用esp_netif_create_default_wifi_sta(); 或 esp_netif_create_default_wifi_ap();。

请参考上述三个接口的初始化代码示例：
- Wi-Fi Station：wifi/getting_started/station/main/station_example_main.c
• Wi-Fi AP：wif/learning/softAP/main/softap_example_main.c
• 以太网：ethernet/basic/main/ethernet_example_main.c

其他 tcpip_adapter API 更换 所有 tcpip_adapter 函数都有对应的 esp-netif，请参考以下章节中的 esp_netif.h 部分，了解更多信息：
  • Setters/Getters
  • DHCP
  • DNS
  • IP address

默认事件处理程序 事件处理程序已从 tcpip_adapter 移至相应驱动程序代码。从应用程序的角度来看，这一变更不会产生任何影响，所有事件仍将以相同的方式处理。请注意，在与 IP 相关的事件处理程序中，应用代码通常以 esp-netif 结构体而非 LwIP 结构体的形式接收 IP 地址。两种结构体兼兼容二进制格式。

打印地址的首选方式如下所示：
```c
ESP_LOGI(TAG, "got ip:\n", IP2STR(&event->ip_info.ip));
```

不建议使用下述方式：
```c
ESP_LOGI(TAG, "got ip:%s\n", ip4addr_ntoa(&event->ip_info.ip));
```

ip4addr_ntoa() 为 LwIP API，因此 esp-netif 还提供了替代函数 esp_ip4addr_ntoa()，然而总得来说仍推荐使用 IP2STR() 这一方法。

IP 地址 推荐使用 esp-netif 定义的 IP 结构。请注意，在启用默认兼容性时，LwIP 结构体仍然可以工作。
  • esp-netif IP address definitions

外设

外设时钟门控 与更新之前相同，外设的时钟仍由驱动处理，用户无需对外设模块的时钟门控进行设置。但是，如果用户想基于组件 hal 和 soc 开发自己的驱动，请注意时钟门控的头文件引用路径已由 driver/periph_ctrl.h 更新为 esp_private/periph_ctrl.h。

RTC 子系统控制 RTC 控制 API 已经从 driver/rtc_ctrl.h 移动到了 esp_private/rtc_ctrl.h。

ADC

ADC 单次模式及连续模式驱动 ADC 单次模式的驱动已更新。
  • 新的驱动位于组件 esp_adc 中，头文件引用路径为 esp_adc/adc_oneshot.h。
  • 旧版驱动仍然可用，其头文件引用路径为 driver/adc.h。

对于 ADC 连续模式驱动，其位置已由组件 driver 更新为 esp_adc。
  • 头文件引用路径由 driver/adc.h 更新为 esp_adc/adc_continuous.h。

但是，引用两种模式的旧版路径 driver/adc.h 会默认触发如下编译警告，可通过配置 Kconfig 选项 CONFIG_ADC_SUPPRESS_DEPRECATED_WARN 关闭该警告。
ADC 校准驱动

- 新的驱动位于组件 esp_adc 中，头文件引用路径为 esp_adc/adc_cali.h 和 esp_adc/adc_cali_scheme.h。
- 旧版驱动仍然可用，其头文件引用路径为 esp_adc_cal.h。如果用户要使用旧版路径，需要将组件 esp_adc 添加到文件 CMakeLists.txt 的组件需求表中。
- 默认情况下，引用路径 esp_adc_cal.h 会默认触发如下编译警告，可通过配置 Kconfig 选项 CONFIG_ADC_CALI.Suppress_Deprecate_Warn 关闭该警告。

### API 更新

- ADC 电源管理 API adc_power_acquire 和 adc_power_release 已被移至 esp_private/adc_share_hw_ctrl.h，用于内部功能。
  - 更新前，由于硬件延迟的工作原理，这两个 API 可以被用户调用。
  - 更新后，ADC 电源管理完全由驱动在内部实现。
  - 如果用户仍需调用这个 API，可以通过引用路径 esp_private/adc_share_hw_ctrl.h 来调用它。
- 更新后，driver/adc2_wifi_private.h 已被移至 esp_private/adc_share_hw_ctrl.h。
- esp_unit_t 中的枚举 ADC_UNIT_BOTH, ADC_UNIT_ALTER 及 ADC_UNIT_MAX 已被删除。
- 由于只有部分芯片支持下列枚举的某些取值，因此将下列枚举删除。如果用户使用了不支持的取值，会造成驱动运行错误。
  - 枚举 ADC_CHANNEL_MAX
  - 枚举 ADC_ATTEN_MAX
  - 枚举 ADC_CONV_UNIT_MAX
- ESP32 中的 API hall_sensor_read 已被删除，因此 ESP32 不再支持霍尔传感器。
- API adc_set_i2s_data_source 和 adc_i2s_mode_init 已被弃用，相关的枚举 adc_i2s_source_t 也被弃用，请使用 esp_adc/adc_continuous.h 进行迁移。

GPIO

- 之前的 Kconfig 选项 RTCIO_SUPPORT_RTC_GPIO_DESC 已被删除，因此数组 rtc_gpio_desc 已不可用，请使用替代数组 rtc_io_desc。
- 更新后，用户调用函数无法再通过读取 GPIO 中断的状态寄存器来获取用于触发中断的 GPIO 管脚的编号。但是，用户可以通过使用回调函数变量来确定该管脚编号。
  - 更新前，GPIO 中断发生时，GPIO 中断状态寄存器调用用户回调函数之后，会被清空。因此，用户可以在回调函数中读取 GPIO 中断状态寄存器，以便确定触发中断的 GPIO 管脚。
  - 但是，在调用回调函数后清空中断状态寄存器可能会导致边缘触发的中断丢失。例如，在调用用户回调函数时，如果某个边缘触发的中断 (re) 被触发，该中断会被清除，并且其注册的用户回调函数还未被处理。
  - 更新后，GPIO 的中断状态寄存器在调用用户回调函数之前被清空。因此，用户无法读取 GPIO 中断状态寄存器来确定哪个管脚触发了中断。但是，用户可以通过回调函数变量来传递被触发的管脚编号。

Sigma-Delta 调制器

- 新驱动中实现了工厂模式，SDM 通道都位于内部通道池中，因此用户无需手动将 SDM 通道配置到 GPIO 管脚。
- SDM 通道会被自动分配。
尽管我们推荐用户使用新的驱动 API，旧版驱动仍然可用，位于头文件引用路径 driver/sigmadelta.h 中。但是，引用 driver/sigmadelta.h 会默认触发如下编译警告，可通过配置 Kconfig 选项 CONFIG_SDM_SUPPRESS_DEPRECATED_WARN 关闭该警告。

The legacy sigma-delta driver is deprecated, please use driver/sdm.h

概念与使用方法上的主要更新如下所示：

**主要概念更新**

- SDM 通道名称已由 sigmadelta_channel_t 更新为 sdm_channel_handle_t，后者为一个不透明指针。
- SDM 通道配置原来存放于 sigmadelta_config_t，现存放于 sdm_config_t。
- 旧版驱动中，用户无需为 SDM 通道设置时钟源。但是在新驱动中，用户需要在 sdm_config_t::clk_src 为 SDM 通道设置合适的时钟源，soc_periph_sdm_clk_src_t 中列出了可用的时钟源。
- 旧版驱动中，用户需要为通道设置 prescale，该参数会影响调制器输出脉冲的频率。在新的驱动中，用户需要使用 sdm_config_t::sample_rate_hz 实现该功能。

**主要使用方法更新**

- 更新前，通道配置由通道分配器 sdm_new_channel() 完成。在新驱动中，只有 duty 可在运行时由 sdm_channel_set_duty() 更新。其他参数如 gpio number, prescale 只能在通道分配时进行设置。
- 在进行下一步通道操作前，用户应通过调用 sdm_channel_enable() 提前使能该通道。该函数有助于管理一些系统级服务，如电源管理。

定时器组驱动 为统一和简化通用定时器的使用，定时器组驱动已更新为 GPTimer。

尽管我们推荐使用新的驱动 API，旧版驱动仍然可用，其头文件引用路径为 driver/timer.h。但是，引用 driver/timer.h 会默认触发如下编译警告，可通过配置 Kconfig 选项 CONFIG_GPTIMER_SUPPRESS_DEPRECATED_WARN 关闭该警告。

legacy timer group driver is deprecated, please migrate to driver/gptimer.h

概念和使用方法上的主要更新如下所示：

**主要概念更新**

- 用于识别定时器的 timer_group_t 和 timer_idx_t 已被删除。在新驱动中，定时器用参数 gptimer_handle_t 表示。
- 更新后，定时器的时钟源由 gptimer_clock_source_t 定义，之前的时钟源参数 timer_src_clk_t 不再使用。
- 更新后，定时器计数方向由 gptimer_count_direction_t 定义，之前的计数方向参数 timer_count_dir_t 不再使用。
- 更新后，仅支持电平触发的中断，timer_intr_t 和 timer_intr_mode_t 不再使用。
- 更新后，通过设置标志位 gptimer_alarm_config_t::auto_reload_on_alarm，可以使能自动生成加裁。timer_autoreload_t 不再使用。

**主要使用方法更新**

- 更新后，通过从 gptimer_new_timer() 创建定时器示例可以初始化定时器。用户可以在 gptimer_config_t 进行一些基本设置，如时钟源，分辨率和计数方向。请注意，无需在驱动安装阶段进行报警事件的特殊设置。
- 更新后，报警事件在 gptimer_set_alarm_action() 中进行设置，参数在 gptimer_alarm_config_t 中进行设置。
• 更新后，通过`gptimer_get_raw_count()`设置计数值，通过`gptimer_set_raw_count()`获取计数值，驱动不会自动将原始数据同步到UTC时间戳。由于时钟器的分辨率已知，用户可以自行转换数据。

• 更新后，如果`gptimer_event_callbacks_t::on_alarm`被设置为有 效的回调函数，驱动程序也会安装中断服务。在回调函数中，用户无需配置底层寄存器，如用于“清除中断状态”、“重新使能事件”的寄存器等。因此，`timer_group_get_intr_status_in_isr`与`timer_group_get_auto_reload_in_isr`这些函数不再使用。

• 更新后，当报警事件发生时，为更新报警配置，用户可以在中断回调中调用`gptimer_set_alarm_action()`，这样报警事件会被重新使能。

• 更新后，如果用户将`gptimer_alarm_config_t::auto_reload_on_alarm`设置为true，报警事件将会被驱动程序使能。

### UART

<table>
<thead>
<tr>
<th>删除/弃用项目</th>
<th>替代</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>uart_isr_register()</td>
<td>无</td>
<td>更新后，UART 中断由驱动处理。</td>
</tr>
<tr>
<td>uart_isr_free()</td>
<td>无</td>
<td>更新后，UART 中断由驱动处理。</td>
</tr>
<tr>
<td>use_ref_tick</td>
<td>uart_config_t::source_clk</td>
<td>选择时钟源。</td>
</tr>
<tr>
<td>uart_enable_pattern_det_baud</td>
<td>UART enable pattern det baud</td>
<td>信号模式检测中断。</td>
</tr>
</tbody>
</table>

### I2C

<table>
<thead>
<tr>
<th>删除/弃用项目</th>
<th>替代</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2c_isr_register()</td>
<td>无</td>
<td>更新后，I2C 中断由驱动处理。</td>
</tr>
<tr>
<td>i2c_isr_deregister()</td>
<td>无</td>
<td>更新后，I2C 中断由驱动处理。</td>
</tr>
<tr>
<td>i2c_opmode_t</td>
<td>无</td>
<td>更新后，该项不再在 esp-idf 中使用。</td>
</tr>
</tbody>
</table>

### SPI

<table>
<thead>
<tr>
<th>删除/弃用项目</th>
<th>替代</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_cal_clock()</td>
<td>spi_get_actual_clock()</td>
<td>获取 SPI 真实的工作频率。</td>
</tr>
</tbody>
</table>

• 内部头文件`spi_common_internal.h`已被移至`esp_private/spi_common_internal.h`。

### SDMMC

<table>
<thead>
<tr>
<th>删除/弃用项目</th>
<th>替代</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdmmc_host_pullup_en()</td>
<td>sdmmc_slot_config_t::flags SDMMC_SLOT_FLAG_INTERNAL_PULLUP</td>
<td>设置标志位 便能内部上拉。</td>
</tr>
</tbody>
</table>

### LEDC

<table>
<thead>
<tr>
<th>删除/弃用项目</th>
<th>替代</th>
<th>备注</th>
</tr>
</thead>
<tbody>
<tr>
<td>ledc_timer_config_t::bit_num</td>
<td>ledc_timer_config_t::duty_resolution</td>
<td>设置占空比分辨率。</td>
</tr>
</tbody>
</table>

脉冲计数器 (PCNT) 驱动 为统一和简化 PCNT 外设，PCNT 驱动已更新，详见 PCNT。

尽管我们推荐使用新的驱动 API，旧版驱动仍然可用，保留在头文件引用路径 `driver/pcnt.h` 中。但是，引用路径 `driver/pcnt.h` 会默认触发如下编译警告，可通过配置 Kconfig 选项`CONFIG_PCNT.Suppress.Deprecate.Warn` 来关闭该警告。
主要概念和使用方法上的更新如下所示:

主要概念更新

- 更新后，pcnt_port_t，pcnt_unit_t 和 pcnt_channel_t 这些用于识别 PCNT 单元和通道的参数已被删除。在新的驱动中，PCNT 单元由参数 pcnt_unit_handle_t 表示，PCNT 通道由参数 pcnt_channel_handle_t 表示，这两个参数都是不透明指针。
- 更新后，不再使用 pcnt_evt_type_t，它们由统一的 观察点事件 表示。在事件回调函数 pcnt_watch_cb_t 中，通过 pcnt_watch_event_data_t 可以分辨不同观察点。
- pcnt_count_mode_t 更名为 pcnt_channel_edge_action_t，pcnt_ctrl_mode_t 更名为 pcnt_channel_level_action_t。

主要使用方法更新

- 更新前，PCNT 的单元配置和通道配置都通过函数 pcnt_unit_config 实现。更新后，PCNT 的单元配置通过工厂 API pcnt_new_unit() 完成，通道配置通过工厂 API pcnt_new_channel() 完成。
  - 只需配置计数范围即可初始化一个 PCNT 单元。更新后，GPIO 管脚分配通过 pcnt_new_channel() 完成。
  - 高低电平控制模式和上升沿/下降沿计数模式分别通过函数 pcnt_channel_set_edge_action() 和 pcnt_channel_set_level_action() 进行设置。
- pcnt_get_counter_value 更新为 pcnt_unit_get_count()。
- pcnt_counter_start 更新为 pcnt_unit_start()。
- pcnt_counter_clear 更新为 pcnt_unit_clear_count()。
- 更新后，pcnt_intr_enable 与 pcnt_intr_disable 已被删除。新的驱动中，通过注册时间回调函数 pcnt_unit_register_event_callbacks() 来使能中断。
- 更新后，pcnt_event_enable 与 pcnt_event_disable 已被删除。新的驱动中，可通过 pcnt_unit_add_watch_point() 和 pcnt_unit_remove_watch_point() 来增加/删除观察点，以使能/停用 PCNT 事件。
- 更新后，pcnt_set_event_value 已被删除。新的驱动中，通过 pcnt_unit_add_watch_point() 增加观察点时，也同时设置了事件的数值。
- 更新后，pcnt_get_event_value 与 pcnt_get_event_status 已被删除。新的驱动中，这些信息存储在 pcnt_watch_event_data_t 的回调函数 pcnt_watch_cb_t 中。
- 更新后，pcnt_isr_register 与 pcnt_isr_unregister 已被删除。不允许注册 ISR 句柄。用户可以通过调用 cpp:func:pcnt_unit_register_event_callbacks 来注册事件回调函数。
- 更新后，pcnt_set_pin 已被删除。新的驱动不再允许在运行时切换 GPIO 管脚。如果用户想切换为其他 GPIO 管脚，可通过 cpp:func:pcnt_del_channel 删除当前的 PCNT 通道，然后通过 cpp:func:pcnt_new_channel 安装新的 GPIO 管脚。
- pcnt_filter_enable, pcnt_filter_disable 与 pcnt_set_filter_value 更新为 pcnt_unit_set_glitch_filter()，同时，pcnt_get_filter_value 已被删除。
- pcnt_set_mode 更新为 pcnt_channel_set_edge_action() 与 pcnt_channel_set_level_action()。
- pcnt_isr_service_install, pcnt_isr_service_unregister, pcnt_isr_handler_add 与 pcnt_isr_handler_remove 更新为 pcnt_unit_register_event_callbacks()。默认的 ISR 句柄已安装在新的驱动中。

**RMT 驱动** 为统一和扩展 RMT 外设的使用，RMT 驱动已更新，详见 RMT transceiver。

尽管我们建议使用新的驱动 API，旧版驱动仍然可用。保留在头文件引用路径 driver/rmt.h 中。但是，引用路径 `driver/rmt.h` 会默认触发如下编译警告，可通过配置 Kconfig 选项 CONFIG_RMT_SUPPRESS_DEPRECATED_WARN 来关闭该警告。
The legacy RMT driver is deprecated, please use driver/rmt_tx.h and/or driver/rmt_
rx.h

主要概念和使用方法更新如下所示：

主要概念更新

- 更新后，用于识别硬件通道的 rmt_channel_t 已删除。在新的驱动中，RMT 通道用参数
  rmt_channel_handle_t 表示，该通道由驱动程序动态分配，而不是由用户指定。
- rmt_item32_t 更新为 rmt_symbol_word_t，以避免在结构体中出现嵌套的共用体。
- 更新后，rmt_mem_t 已被删除，因为我们不只允许用户直接访问 RMT 内存块（即 RMTMEM）。直接
  访问 RMTMEM 没有意义，反而会引发错误，特别是当 RMT 通道与 DMA 通道相连时。
- 更新后，由 rmt_mem_owner_t 由驱动控制，而不是用户，因此 rmt_mem_owner_t 已被删除。
- rmt_source_clk_t 更新为 rmt_clock_source_t，后者不支持二进制兼容。
- 更新后，rmt_data_mode_t 已被删除，RMT 内存访问模式配置为始终使用 Non-FIFO 和 DMA 模式。
- 更新后，由于驱动有独立的发送和接收通道装术函数，因此 rmt_mode_t 已被删除。
- 更新后，rmt_idle_level_t 已被删除，在 rmt_transmit_config_t::eot_level 中可为发送
  通道设置空闲状态电平。
- 更新后，rmt_carrier_level_t 被删除，可在 rmt_carrier_config_t::polarity_active_low
  设置载波极性。
- 更新后，rmt_channel_status_t 与 rmt_channel_status_result_t 已被删除，不再使用。
- 通过 RMT 通道发送并不需要用户提供 RMT 符号，但是用户需要提供一个 RMT 编码器用来告知驱
  动如何将用户数据转换成 RMT 符号。

主要使用方法更新

- 更新后，分别通过 rmt_new_tx_channel() 和 rmt_new_rx_channel() 安装发送通道和接收
  通道。
- 更新后，rmt_set_clk_div 和 rmt_get_clk_div 已被删除。通道时钟配置只能在通道安装时
  完成。
- 更新后，rmt_set_rx_idle_thresh 和 rmt_get_rx_idle_thresh 已被删除。新驱动中，接
  收通道的空闲状态阈值定义为 rmt_receive_config_t::signal_range_max_ns。
- 新 驱 动 中， 内 存 块 的 数 量 由 rmt_tx_channel_config_t::mem_block_symbols
  与 rmt_rx_channel_config_t::mem_block_symbols 决定。
- 更新后，rmt_set_tx_carrier 已被删除。新驱动使用 rmt_apply_carrier() 来设置载波波
  动。
- 更新后，rmt_set_mem_pd 和 rmt_get_mem_pd 已被删除，驱动程序自动调整内存的功率。
- 更新后，rmt_memory_rx_rst, rmt_tx_memory_reset 和 rmt_rx_memory_reset 已被删
  除，驱动程序自动进行内存重置。
- 更新后，rmt_tx_start 和 rmt_rx_start 被合并为函数 rmt_enable()，该函数同时适用于
  发射通道和接收通道。
- 更新后，rmt_tx_stop 和 rmt_rx_stop 被合并为函数 rmt_disable()，该函数同时适用于发
  射通道和接收通道。
- 更新后，rmt_set_memory_owner 和 rmt_get_memory_owner 已被删除，驱动程序自动添加
  RMT 内存保护。
- 更新后，rmt_set_tx_loop_mode 和 rmt_get_tx_loop_mode 已被删除。新驱动在，
  rmt_transmit_config_t::loop_count 中设置循环模式。
- 更新后，rmt_set_source_clk 和 rmt_get_source_clk 已被删除。仅能在通道安装时通过
  rmt_tx_channel_config_t::clk_src 和 rmt_rx_channel_config_t::clk_src 设置时钟源。
- 更新后，rmt_set_rx_filter 已被删除。新驱动中，过滤阈值定义为
  rmt_receive_config_t::signal_range_min_ns。
- 更新后，rmt_set_idle_level 和 rmt_get_idle_level 已被删除，可在
  rmt_transmit_config_t::eot_level 中设置发送通道的空闲状态电平。
Chapter 5. 迁移指南

- 更新后，rmt_set_rx_intr_en, rmt_set_err_intr_en, rmt_set_tx_intr_en, rmt_set_tx_thr_intr_en 和 rmt_set_rx_thr_intr_en 已被删除。新驱动不允许用户在用户端开启/关闭中断，而是提供了回调函数。
- 更新后，rmt_set_gpio 和 rmt_set_pin 已被删除。新驱动不支持运行时动态切换 GPIO 管脚。
- 更新后，rmt_config 已被删除。新驱动中，基础配置在通道安装阶段完成。
- 更新后，rmt_isr_register 和 rmt_isr_deregister 已被删除，驱动程序负责分配中断。
- rmt_driver_install 更新为 rmt_new_tx_channel() 中 rmt_new_rx_channel()。
- rmt_driver_uninstall 更新为 rmt_del_channel()。
- 更新后，rmt_fill_tx_items, rmt_write_items 和 rmt_write_sample 已被删除。新驱动中，用户需要提供一个编码器用来将用户数据“翻译”为 RMT 码。
- 更新后，由于用户可以通过 rmt_tx_channel_config_t::resolution_hz 配置通道的时钟分辨率，rmt_get_counter_clock 已被删除。
- rmt_wait_tx_done 更新为 rmt_tx_channel_config_t::resolution_hz 和 rmt_set_pin 重复。
- 更新后，rmt_set_gpio 和 rmt_set_pin 已被删除。新驱动中，rmt_set_tx_intr_en 已被删除，用户需提供一个编码器用来将用户数据“翻译”为 RMT 码。
- 更新后，由于用户可以通过 rmt_tx_channel_config_t::resolution_hz 配置通道的时钟分辨率，rmt_get_counter_clock 已被删除。
- rmt_wait_tx_done 更新为 rmt_tx_channel_config_t::resolution_hz 和 rmt_set_pin 重复。

LCD

- LCD 面板的初始化流程也有一些更新。更新后，esp_lcd_panel_init() 不再会自动打开显示器。用户需要调用 esp_lcd_panel_disp_on_off() 手动打开显示器，请注意，打开显示器与打开背光是不同的。更新后，打开屏幕前，用户可以烧录一个预定义的图案，这可以避免开机复位后屏幕上的随机噪音。
- 更新后，esp_lcd_panel_disp_off() 已被弃用，请使用 esp_lcd_panel_disp_on_off() 作为替代。
- 更新后，dc_as_cmd_phase 已被删除，SPI LCD 驱动不再支持 9-bit 的 SPI LCD，请使用专用的 GPIO 管脚来控制 LCD D/C 线。
- 更新后，用于注册 RGB 面板的事件回调函数已从 esp_lcd_rgb_panel_config_t 更新为单独的 API esp_lcd_rgb_panel_register_event_callbacks()。但是，事件回调签名仍保持不变。
- 更新后，esp_lcd_rgb_panel_config_t 中的标志位 relax_on_idle 被重命名为 esp_lcd_rgb_panel_config_t::refresh_on_demand，后者虽表达了同样的含义，但是其命名更有意义。
- 更新后，如果创建 RGB LCD 时，标志位 refresh_on_demand 使能，驱动不会在 esp_lcd_panel_draw_bitmap() 中进行刷新，用户需要调用 esp_lcd_rgb_panel_refresh() 来刷新屏幕。
- 更新后，esp_lcd_color_space_t 已被弃用，请使用 lcd_color_space_t 来描述色彩空间，使用 lcd_color_rgb_endian_t 来描述 RGB 颜色的排列顺序。

MCPWM

MCPWM 支持的功能旨在方便用户使用。同时，旧版驱动已被弃用。

新驱动中，每个 MCPWM 子模块相互独立，用户可以自由进行资源选择。

尽管我们推荐使用新的驱动，API 和旧版驱动仍然可用，其包含路径为 driver/mcpwm.h。但是，使用旧版驱动会默认触发如下编译警告，可以通过配置 Kconfig 选项 CONFIG_MCPWM_SUPPRESS_DEPRECATE_WARN 来关闭该警告。
Chapter 5. 迁移指南

主要概念和使用方法上的更新如下所示:

主要概念更新  更新后，MCPWM 驱动是面向对象的，大多数 MCPWM 子模块都有一个与之相关的驱动对象。驱动对象是由工厂函数创建的，如 `mcpwm_new_timer()`。IO 控制函数总是需要对象句柄。

旧版驱动有一个不恰当的假设，即 MCPWM 运算器应连接到不同的 MCPWM 定时器上，但是，硬件上并没有这样的限制。新驱动中，同一个 MCPWM 定时器可以连接多个运算器，这样运算器可以获得最佳的同步性能。

更新前，驱动将生成 PWM 波形的方法预设为所谓的 `mcpwm_duty_type_t`，但是，列出的占空比模式远远不够。类似的，旧版驱动有一些预设的 `mcpwm_deadtime_type_t` 也没有包含所有的使用场景。更重要的是，用户通常会被占空比模式和死区时间模式的名称所迷惑。更新后，驱动没有这些限制，但是用户必须从头开始构建发生器的行为。

在旧版驱动中，通过 GPIO 管脚，软件和其他定时器模块同步 MCPWM 定时器的方法并不统一。这增加了用户的学习成本，因此新驱动统一了同步 API。

旧版驱动混淆了“故障检测器”和“故障处理器”的概念。这让用户对 API 感到非常困惑。新驱动中，故障对象只代表一个故障源，而我们引入了一个新概念，制动器，来表示故障处理器。而且，新驱动支持软件故障。

旧版驱动只为获取子模块提供了回调函数，而新驱动为 MCPWM 子模块提供多种回调函数，如停止定时器，比较匹配，故障进入，紧急停止等。

- 更新后，不再使用 `mcpwm_io_signals_t` 和 `mcpwm_pin_config_t`，GPIO 管脚配置被移至子模块的配置结构中。
- 更新后，不再使用 `mcpwm_timer_t` 和 `mcpwm_generator_t`，定时器和发生器分别用 `mcpwm_timer_handle_t` 和 `cpp: type=mcpwm_gen_handle_t` 表示。
- 更新后，不再使用 `mcpwm_fault_signal_t` 和 `mcpwm_sync_signal_t`，故障和同步源分别用 `mcpwm_fault_handle_t` 和 `cpp: type=mcpwm_sync_handle_t` 表示。
- 更新后，不再使用 `mcpwm_capture_signal_t`，获取通道用 `mcpwm_cap_channel_handle_t` 表示。

主要使用方法更新

- `mcpwm_gpio_init` 和 `mcpwm_set_pin`：GPIO 管脚配置在子模块配置中完成，例如在 `mcpwm_generator_config_t::gen_gpio_num` 中设置 PWM GPIO 管脚。
- `mcpwm_init`：为得到预期的 PWM 波形，用户需要至少分配一个 MCPWM 定时器和 MCPWM 运算器，然后通过调用 `mcpwm_operator_connect_timer()` 将二者连接起来。然后，用户需要调用如 `cpp: func=mcpwm_generator_set_actions_on_timer_event`，`mcpwm_generator_set_actions_on_compare_event()` 来设置发生器对不同事件的动作。
- `mcpwm_group_set_resolution`：新驱动中，群组分辨率固定为最大值。通常为 80 MHz。
- `mcpwm_timer_set_resolution`：MCPWM 定时器的分辨率在 `mcpwm_timer_config_t::resolution_hz` 中进行设置。
- `mcpwm_set_frequency`：PWM 频率由 `mcpwm_timer_config_t::resolution_hz` 和 `mcpwm_timer_config_t::count_mode` 和 `cpp: member=mcpwm_timer_config_t::period_ticks` 决定。
- `mcpwm_set_duty`：为设置 PWM 占空比，用户应调用 `mcpwm_comparator_set_compare_value()` 来改变比较器的阈值。
- `mcpwm_set_duty_type`：新驱动中没有预设的占空比模式，通过设置不同的发生器行为，如 `mcpwm_generator_set_actions_on_timer_event()`，来配置占空比模式。
- `mcpwm_set_signal_high` 和 `mcpwm_set_signal_low` 更新为 `mcpwm_generator_set_signal_high` 和 `mcpwm_generator_set_signal_low`，新驱动中，这是通过为发生器设置力作用来实现的，而不是在后台将占空比改为 0% 或 100%。
- `mcpwm_start` 和 `mcpwm_stop` 更新为 `mcpwm_timer_start_stop()`。用户可以用更多的模式来启动和停止 MCPWM 定时器，详见 `mcpwm_timer_start_stop_cmd_t`。
- `mcpwm_carrier_init` 更新为 `mcpwm_operator_apply_carrier()`。

legacy MCPWM driver is deprecated, please migrate to the new driver (include_driver/mcpwm_prelude.h)
Chapter 5. 迁移指南

- mcpwm_carrier_enable 与 mcpwm_carrier_disable：通过检查载波设置结构 mcpwm_carrier_config_t 是否为空来决定使能和禁止载波输出。
- mcpwm_carrier_set_period 更新为 mcpwm_carrier_config_t::frequency_hz。
- mcpwm_carrier_set_duty_cycle 更新为 mcpwm_carrier_config_t::duty_cycle。
- mcpwm_carrier_oneshot_mode_enable 更新为 mcpwm_carrier_config_t::first_pulse_duration_us。
- 更新后，mcpwm_carrier_oneshot_mode_disable 被删除。硬件不支持停用第一个载波脉冲（即一次性脉冲）。
- mcpwm_carrier_output_invert 更新为 mcpwm_carrier_config_t::invert_before_modulate 和 mcpwm_carrier_config_t::invert_after_modulate。
- mcpwm_deadtime_enable 与 mcpwm_deadtime_disable 更新为 mcpwm_generator_set_dead_time()。
- mcpwm_fault_init 更新为 mcpwm_new_gpio_fault()。
- mcpwm_fault_set_cyc_mode 与 mcpwm_fault_set_cyc_mode 更新为 mcpwm_operator_set_brake_on_fault() 与 mcpwm_generator_set_actions_on_brake_event()。
- 由于 mcpwm_capture_enable 与 mcpwm_capture_enable_channel() 重复，因此在更新后被删除。
- 由于 mcpwm_capture_disable 与 mcpwm_capture_disable_channel() 重复，因此在更新后被删除。
- mcpwm_capture_enable_channel 与 mcpwm_capture_disable_channel 重复为 mcpwm_capture_enable_channel() 与 mcpwm_capture_disable_channel()。
- mcpwm_capture_signal_get_value 与 mcpwm_capture_signal_get_edge：通过 mcpwm_capture_event_data_t，获取事件回调函数中提供了计时器的数值和边缘电平。只有获取事件发生时，获取数据才有意义，提供单一的 API 来获取捕获数据是没有意义的。
- 由于 mcpwm_sync_enable 与 mcpwm_sync_disable 重复，因此更新后被删除。
- mcpwm_sync_config 更新为 mcpwm_timer_set_sync_phase_config_t::sync_src 设置为 NULL。
- mcpwm_set_timer_sync_output 更新为 mcpwm_new_timer_sync_src()。
- mcpwm_timer_trigger_soft_sync 更新为 mcpwm_soft_sync_activate()。
- mcpwm_syncinvert_gpio_synchro 与设置 mcpwm_gpio_sync_config_t::active_neg 功能相同。
- 更新后，mcpwm_isr_register 已被删除。用户可以注册不同的事件回调函数来替换其功能，例如，可以使用 mcpwm_capture_channel_register_event_callbacks() 注册获取事件注册函数。

I2S 驱动 旧版 I2S 驱动在支持 ESP32-C3 和 ESP32-S3 新功能时暴露了很多缺点，为解决这些缺点，I2S 驱动已更新（请参考 doc/I2S Driver <.../api-reference/ peripherals/i2s>)。用户可以通过引用不同 I2S 模式对应的头文件来使用新版驱动的 API，如 driver/include/driver/i2s_std.h, driver/include/driver/i2s_pdm.h 以及 driver/include/driver/i2s_tdm.h。

为保证前向兼容，旧版驱动的 API 仍然在 driver/deprecated/driver/i2s.h 中可用。但使用旧版 API 会触发编译警告，该警告可通过配置 Kconfig 选项 CONFIG_I2S_SUPPRESS_DEPRECATED_WARN 来关闭。

以下是更新后的 I2S 文件概况。

主要概念更新

独立的发送通道和接收通道 更新后，I2S 驱动的最小控制单元是发送/接收通道，而不是整个 I2S 控制器（控制器包括多个通道）。
- 用户可以分别控制同一个 I2S 控制器的发送通道和接收通道，即可以通过配置实现分别开启和关闭发送通道和接收通道。
- i2s-chan_handle_t 句柄类型用于唯一地识别 I2S 通道。所有的 API 都需要该通道句柄，用户需要对这些通道句柄进行维护。
- 对于 ESP32-C3 和 ESP32-S3，同一个控制器中的发送通道和接收通道可以配置为不同的时钟或不同的模式。
- 但是对于 ESP32 和 ESP32-S2，同一个控制器中的发送通道和接收通道共享某些硬件资源。因此，配置可能会造成一个通道影响同一个控制器中的另一个通道。
通过将 `i2s_port_t::I2S_NUM_AUTO` 设置为 I2S 端口 ID，驱动会搜索可用的发送/接收通道，之后通道会被自动注册到可用的 I2S 控制器上。但是，驱动仍然支持将通道注册到一个特定的端口上。

为区分发送/接收通道和声音通道，在更新后的驱动中，“通道 (channel)”一词仅代表发送/接收通道，用“声道 (slot)”来表示声音通道。

I2S 模式分类 I2S 通信模式包括以下三种模式，请注意：

- **标准模式**：标准模式通常包括两个声道，支持 Philips，MSB 和 PCM（短帧同步）格式，详见 `driver/include/driver/i2s_std.h`。
- **PDM 模式**：PDM 模式仅支持两个声道，16 bit 数据位宽，但是 PDM TX 和 PDM RX 的配置略有不同。对于 PDM TX，采样率可通过 `i2s_pdm_rx_clk_config_t::sample_rate` 进行设置，其时钟频率取决于上采样的配置。对于 PDM RX，采样率可通过 `i2s_pdm_rx_clk_config_t::sample_rate` 进行设置，其时钟频率取决于下采样的配置，详见 `driver/include/driver/i2s_pdm.h`。
- **TDM 模式**：TDM 模式可支持高达 16 声道，该模式可在 Philips，MSB，PCM（短帧同步）和 PCM（长帧同步）格式下，详见 `driver/include/driver/i2s_tdm.h`。

在某个模式下分配新通道时，必须通过相应的函数初始化这个通道。我们强烈建议使用辅助宏来生成默认配置，以避免默认值被改动。

独立的声道配置和时钟配置 可以单独进行声道配置和时钟配置。

- 通过调用 `i2s_channel_init_std_mode()`，`i2s_channel_init_pdm_rx_mode()`，`i2s_channel_init_pdm_tx_mode()` 或 `cppfunc::i2s_channel_init_tdm_mode` 初始化声道/时钟/GPIO 管脚配置。
- 通过调用 `i2s_channel_reconfig_std_slot()`，`i2s_channel_reconfig_pdm_rx_slot()` 或 `i2s_channel_reconfig_pdm_rx_slot()` 可以在初始化之后改变声道配置。
- 通过调用 `i2s_channel_reconfig_std_slot()`，`i2s_channel_reconfig_pdm_rx_slot()` 或 `i2s_channel_reconfig_tdm_slot()` 可以在初始化之后改变时钟配置。
- 通过调用 `i2s_channel_reconfig_std_gpio()`，`i2s_channel_reconfig_pdm_rx_gpio()`，`i2s_channel_reconfig_pdm_rx_gpio()` 或 `i2s_channel_reconfig_tdm_gpio()` 可以在初始化之后改变 GPIO 管脚配置。
更新后，I2S 驱动利用状态和状态机避免在错误状态下调用 API。
更新后，ADC 和 DAC 模式已被删除，只有它们各自专用的驱动及 I2S 旧版驱动还支持这两种模式。

**主要使用方法更新** 请参考以下步骤使用更新后的 I2S 驱动:

1. 通过调用 `i2s_new_channel()` 来获取通道句柄。我们应在这个步骤中指定外设为主机还是从机以及 I2S 端口。此外，驱动负责生成发送通道或接收通道的句柄。不需要同时输入发送通道和接收通道句柄，但需要输入至少一个句柄。输入两个句柄时，驱动会工作在双工模式。在同一端口上，发送通道和接收通道同时可用，并且共享 MCLK，BCLK 和 WS 信号。如果只输入了发送通道句柄或接收通道句柄，该通道只能工作在单工模式。
2. 通过调用 `i2s_channel_init_std_mode()`, `i2s_channel_init_pdm_rx_mode()`, `i2s_channel_init_pdm_tx_mode()` 或 `i2s_channel_init_tdm_mode()` 将通道初始化为指定模式。进行相应的声道、时钟和 GPIO 管脚配置。
3. （可选）通过调用 `i2s_channel_register_event_callback()` 注册 ISR 事件回调函数。I2S 事件由回调函数同步接收，而不是从事件队列中异步接收。
4. 通过调用 `i2s_channel_enable()` 来开启 I2S 通道的硬件资源。在更新后的驱动中，I2S 在安装后不会再自动开启。用户需要确定通道是否已经开启。
5. 分别通过 `i2s_channel_read()` 和 `i2s_channel_write()` 来读取和写入数据。当然，在 `i2s_channel_read()` 中只能输入接收通道句柄，在 `i2s_channel_write()` 中只能输入发送通道句柄。
6. （可选）通过相应的 ‘reconfig’ 函数可以更改声道、时钟和 GPIO 管脚配置，但是更改配置前必须调用 `i2s_channel_disable()`。
7. 通过调用 `i2s_channel_disable()` 可以停止使用 I2S 通道的硬件资源。
8. 不再使用某通道时，通过调用 `i2s_del_channel()` 可以删除和释放该通道资源，但是删除之前必须先停用该通道。

### 用于访问寄存器的宏
更新后，所有用于访问寄存器的宏都可以作为表达式来使用，所以以下命令是允许的:

```c
uint32_t val = REG_SET_BITS(reg, mask);
```

在 ESP-IDF v5.0 中，用于写入或读取-修改-写入寄存器的宏不能再作为表达式使用，而只能作为语句使用。这适用于以下宏: `REG_WRITE`, `REG_SET_BIT`, `REG_CLR_BIT`, `REG_SET_BITS`, `REG_SET_FIELD`, `WRITE_PERI_REG`, `CLEAR_PERI_REG_MASK`, `SET_PERI_REG_MASK`, `SET_PERI_REG_BITS`。

为存储要写入寄存器的值，请按以下步骤完成操作:

```c
uint32_t new_val = REG_READ(reg) | mask;
REG_WRITE(reg, new_val);  
```

要获得修改后的寄存器的值（该值可能与写入的值不同），要增加一个显示的读取命令:

```c
REG_SET_BITS(reg, mask);
uint32_t new_val = REG_READ(reg);  
```

### 协议

**Mbed TLS** 在 ESP-IDF v5.0 版本中，Mbed TLS 已从 v2.x 版本更新到 v3.1.0 版本。

更多有关 Mbed TLS 从 v2.x 版本迁移到 v3.0 或更高版本的详细信息，请参考 官方指南。

### 重大更新（概述）
Chapter 5. 迁移指南

增加私有结构体字段数量

- 不再支持直接访问公共头文件中声明的结构体（struct 类型）字段。
- 当前版本下，访问公共头文件中声明的结构体字段需要使用特定的访问函数（getter/setter）。另外，也可以用 MBEDTLS_PRIVATE 宏暂时代替，但不建议使用此种方法。
- 更多详细信息，请参考 官方指南。

SSL

- 不再支持 TLS 1.0、TLS 1.1 和 DTLS 1.0
- 不再支持 SSL 3.0

移除密钥模块中的废弃函数

- 更新了 MD、SHA、RIPEMD、RNG、HMAC 模块相关的函数 mbedtls_* _ret() 的返回值，并将其重新命名，以取代未附加 chaired 的相关函数。
- 更多详细信息，请参考 官方指南。

废弃配置选项

以下为在此次更新中废弃的重要配置选项。与以下配置有关或是依赖于下表配置的相关配置也已相应废弃。

- MBEDTLS_SSL_PROTO_SSL3: 原用于支持 SSL 3.0
- MBEDTLS_SSL_PROTO_TLS1: 原用于支持 TLS 1.0
- MBEDTLS_SSL_PROTO_TLS1_1: 原用于支持 TLS 1.1
- MBEDTLS_SSL_PROTO_DTLS: 原用于支持 DTLS 1.1（当前版本仅支持 DTLS 1.2）
- MBEDTLS_DES_C: 原用于支持 3DES 密码套件
- MBEDTLS_RC4_MODE: 原用于支持基于 RC4 的密码套件

备注：上述仅列出了可通过 idf.py menuconfig 配置的主要选项。更多有关废弃选项的信息，请参考 官方指南。

其他更新

禁用 Diffie-Hellman 密码交换模式

为避免 安全风险，当前版本已默认禁用 Diffie-Hellman 密码交换模式。以下为相应的禁用配置项：

- MBEDTLS_DH_C: 原用于支持 Diffie-Hellman-Merkle 模块
- MBEDTLS_KEY_EXCHANGE_DHE_PSK: 原用于支持 Diffie-Hellman 预共享密钥 (PSK) TLS 认证模式
- MBEDTLS_KEY_EXCHANGE_DHE_RSA: 原用于支持有前缀的密码套件 TLS-DHE-RSA-WITH-

备注：在信号交换的初始步骤（即 client_hello）中，服务器会在客户端提供的列表中选择一个密码。由于 DHE_PSK/DHE_RSA 密码已在本次更新中禁用，服务器将退回到一个替代密码。在特别情况下，服务器按不支持任何其他的代码，此时，初始步骤将失败。若要检索服务器所支持的密码列表，需要首先在客户端使用特定的密码连接服务器，可以使用 sslscan 等工具完成连接。

从 X509 库中移除 certs 模块

- mbeds 3.1 不再支持 mbedts/certs.h 头文件。大多数应用程序支持从包含列表中安全删除该头文件。

对 esp_crt_bundle_set API 的重大更新

- 更新后，调用 esp_crt_bundle_set() API 需要一个额外的参数 bundle_size。该 API 的返回类型也从 void 变为了 esp_err_t.
对 esp_ds_rsa_sign API 的重大更新

- 更新后，调用 esp_ds_rsa_sign() API 无需再使用参数 mode。

HTTPS 服务器

重大更新（概述） 更新 httpd_ssl_config_t 结构体中持有不同证书的变量名。

- httpd_ssl_config::servercert: 原 cacert_pem
- httpd_ssl_config::servercert_len: 原 cacert_len
- httpd_ssl_config::cacert_pem: 原 client_verify_cert_pem
- httpd_ssl_config::cacert_len: 原 client_verify_cert_len

httpd_ssl_stop() API 的返回类型从 void 变为了 esp_err_t。

ESP HTTPS OTA

重大更新（概述）

- 函数 esp_https_ota() 现需以指向 esp_https_ota_config_t 的指针作为参数，而非之前的指向 esp_http_client_config_t 的指针。

ESP-TLS

重大更新（概述）

私有化 esp_tls_t 结构体 更新后，esp_tls_t 已完全私有化，用户无法直接访问其内部结构。之前需要通过 ESP-TLS 句柄获得的必要数据，现在可由对应的 getter/setter 函数获取。如需特定功能的 getter/setter 函数，请在 ESP-IDF 的 Issue 板块 提出。

下列为新增的 getter/setter 函数：

- esp_tls_get_ssl_context(): 从 ESP-TLS 句柄获取底层 ssl 栈的 ssl 上下文。

废弃函数及推荐的替代函数 下表总结了在 ESP-IDF v5.0 中废弃的函数以及相应的替代函数。

<table>
<thead>
<tr>
<th>废弃函数</th>
<th>替代函数</th>
</tr>
</thead>
<tbody>
<tr>
<td>esp_tls_conn_new()</td>
<td>esp_tls_conn_new_sync()</td>
</tr>
<tr>
<td>esp_tls_conn_delete()</td>
<td>esp_tls_conn_destroy()</td>
</tr>
</tbody>
</table>

- 函数 esp_tls_conn_http_new() 现已废弃，已使用替代函数 esp_tls_conn_http_new_sync() (或其异步函数 esp_tls_conn_http_new_async())。请注意，使用替代函数时，需要额外的参数 esp_tls_t，此参数必须首先通过 esp_tls_init() 函数进行初始化。

HTTP 服务器

重大更新（概述）

- esp_http_server 现不再支持 http_server.h 头文件。请使用 esp_http_server.h。
ESP HTTP 客户端

重大更新（概述）

- 函数 esp_http_client_read() 和 esp_http_client_fetch_headers() 现在会返回额外的返回值 -ESP_ERR_HTTP_EAGAIN 用于处理超时错误。即数据准备好前就已调用超时的情况。

TCP 传输

重大更新（概述）

- 更新后，出现连接超时的情况时，函数 esp_transport_read() 将返回 0，对其他错误则返回 < 0。请参阅 esp_tcp_transport_err_t，查看所有可能的返回值。

MQTT 客户端

重大更新（概述）

- esp_mqtt_client_config_t 的所有字段都分组存放在子结构体中。以下为较为常用的配置选项：
  - 通过 esp_mqtt_client_config_t::broker::address::uri 配置 MQTT Broker
  - 通过 esp_mqtt_client_config_t::broker::verification 配置 MQTT Broker 身份验证的相关安全问题
  - 通过 esp_mqtt_client_config_t::credentials::username 配置客户端用户名

ESP-Modbus

重大更新（概述） 本次更新从 ESP-IDF 中移除了组件 freemodbus，该组件已作为一个独立组件受到支持。可前往如下的独立仓库，查看更多有关 ESP-Modbus 的信息：

- GitHub 中的 ESP-Modbus 组件

在新版应用程序中，main 组件文件夹应包括组件管理器清单文件 idf.component.yml，如下所示：

```
dependencies:
espressif/esp-modbus:
  version: "^1.0"
```

可以前往 组件管理器注册表 找到 ESP-Modbus 组件。更多有关如何设置组件管理器的信息，请参考 组件管理器文档。

对于使用 ESP-IDF v4.x 及以后版本的应用程序，需要通过添加组件管理器清单文件 idf.component.yml 拉取新版 ESP-Modbus 组件。同时，在编译时，应去掉已过时的 freemodbus 组件。此项操作可通过项目 CMakeLists.txt 中的以下语句实现：

```
set(EXCLUDE_COMPONENTS freemodbus)
```

配置
Protocomm  protocomm_set_security() API 中的 pop 字段现已弃用。请使用 sec_params 字段来代替 pop。此参数应包含所使用的协议版本所要求的结构（包括安全参数）。

例如，当使用安全版本 2 时，sec_params 参数应包含指向protocomm_security2_params_t 类型结构的指针。

Wi-Fi 配置

- wifi_prov_mgr_start_provisioning() API 中的 pop 字段现已弃用。为了向后兼容，在使用安全版本 1 时，pop 仍可以作为字符串传递。但在使用安全版本 2 时，请使用wifi_prov_sec_params 字段来代替 pop。此参数应包含所使用的协议版本所要求的结构（包括安全参数）。例如，当使用安全版本 2 时，wifi_prov_sec_params 参数应包含指向wifi_prov_security2_params_t 结构体类型的指针。对于安全版本 1，该 API 的行为和使用方式保持不变。

ESP 本地控制 esp_local_ctrl_proto_sec_cfg_t API 中的 pop 字段现已弃用。请使用sec_params 字段来代替 pop。此参数应包含所使用的协议版本所要求的结构（包括安全参数）。

例如，当使用安全版本 2 时，sec_params 字段应包含指向esp_local_ctrl_security2_params_t 类型结构的指针。

从 ESP-IDF 中移出或弃用的组件

移至 IDF Component Registry 的组件 以下组件已经从 ESP-IDF 中迁出至 IDF Component Registry：

- libsodium
- cbor
- jsmn
- esp_modem
- nghttp
- mdns
- esp_websocket_client
- asio
- freemodbus
- sh2lib
- expat
- coap
- esp-cryptoauthlib
- qrcode
- tjpgd
- esp_serial_slave_link
- tinyusb

备注：请注意，http 解析功能以前属于 nghttp 组件一部分，但现在属于 http_parser 组件。

可使用 idf.py add-dependency 命令安装以上组件。

例如，要安装 X.Y 版本的 libsodium 组件，请运行: idf.py add-dependency libsodium==X.Y。

根据 semver 规则安装与 X.Y 兼容的最新版本 libsodium 组件，请运行 idf.py add-dependency

libodium=X.Y。

可前往 https://components.espressif.com 查询每个组件有哪些版本，按名称搜索该组件，组件页面上会列出所有版本。
弃用的组件  IDF v4.x 版本中已不再使用以下组件，这些组件已弃用：

- tcpip_adapter，可使用 ESP-NETIF 组件代替，具体可参考 TCP/IP 配置。

备注：不再支持 OpenSSL-API 组件，IDF Component Registry 中也没有该组件。请直接使用 ESP-TLS 或 mbedtls API。

备注：不再支持 esp_adc_cal 组件，新的 adc 校准驱动在 esp_adc 组件中。旧版 adc 校准驱动已被迁移进 esp_adc 组件中。要使用旧版 esp_adc_cal 驱动接口，你应该在 CMakeLists.txt 文件的组件依赖列表中增加 esp_adc。更多细节请查看 Peripherals Migration Guide。

版本更新后无需目标组件，因此以下目标组件也已经从 ESP-IDF 中删除：

- esp32
- esp32s2
- esp32s3
- esp32c2
- esp32c3
- esp32h4

存储

分区 API 的新组件 非兼容性更新：所有的分区 API 代码都已迁移到新组件 esp_partition 中。如需查看所有受影响的函数及数据类型，请参见头文件 esp_partition.h。

在以前，这些 API 函数和数据类型属于 spi_flash 组件，因此，在现有的应用程序中或将依赖 spi_flash，这也意味着在直接使用 esp_partition_* API/数据类型时，可能会导致构建过程失败（比如，在出现 #include "esp_partition.h" 的行中报错 fatal error: esp_partition.h: No such file or directory）。如果遇到类似问题，请按以下步骤更新项目中的 CMakeLists.txt 文件：

原有的依赖性设置:

```c
idf_component_register(...
    REQUIRES spi_flash)
```

更新后的依赖性设置:

```c
idf_component_register(...
    REQUIRES spi_flash esp_partition)
```

备注：请根据项目的实际情况，更新相应的 REQUIRES 或是 PRIV_REQUIRES 部分。上述代码片段仅为范例。

如果问题仍未解决，请联系我们，我们将协助您进行代码迁移。

SDMMC/SDSPI 用户现可通过 sdmmc_host.t_max_freq_khz 将 SDMMC/SDSPI 接口上的 SD 卡频率配置为特定值，不再局限于之前的 SDMMC_FREQ_PROBING (400 kHz)、SDMMC_FREQ_DEFAULT (20 MHz) 或是 SDMMC_FREQ_HIGHSPEED (40 MHz)。此前，如果用户配置了上述三个给定频率之外的值，用户所选频率将自动调整为与其最为接近的给定值。

更新后，底层驱动将计算与用户配置的特定值最为接近的合适频率。相对于枚举项选择，该频率现由可用的分频器提供。不过，如果尚未更新现有的应用代码，可能会导致与 SD 卡的通信过程出现问题。如发现上述问题，请尝试配置与期望值接近的不同频率，直到找到合适的频率。如需查看底层驱动的计算结果以及实际应用的频率，请使用 void sdmmc_card_print_info(FILE* stream, const sdmmc_card_t* card) 函数。
FatFs

FatFs 已更新至 v0.14，f_mkfs() 函数签名也已变更。新签名为 FRESULT f_mkfs (const TCHAR* path, const MKFS_PARM* opt, void* work, UINT len);，使用 MKFS_PARM 结构体作为第二个实参。

分区表
分区表生成器不再支持未对齐的分区。生成分区表时，ESP-IDF 将只接受偏移量与 4 KB 对齐的分区。此变更仅影响新生成的分区表，不影响读写现有分区。

VFS
esp_vfs_semihost_register() 函数签名有所更改:

- 新签名为了 esp_err_t esp_vfs_semihost_register(const char* base_path);
- 旧签名的 host_path 参数不再存在，请使用 OpenOCD 命令 ESP_SEMIHOST_BASEDIR 设置主机上的完整路径。

函数签名更改
以下函数现将返回 esp_err_t，而非 void 或 nvs_iterator_t。此前，当参数无效或内部出现问题时，这些函数将 assert() 反回 nullptr，通过返回 esp_err_t，您将获得更加实用的错误报告。

- nvs_entry_find()
- nvs_entry_next()
- nvs_entry_info()

由于 esp_err_t 返回类型的更改，nvs_entry_find() 和 nvs_entry_next() 的使用模式也发生了变化。上述函数现均通过参数修改迭代器，而非返回一个迭代器。

迭代 NVS 区分的旧编程模式如下所示:

```c
nvs_iterator_t it = nvs_entry_find(<nvs_partition_name>, <namespace>, NVS_TYPE_ ANY);
while (it != NULL) {
    nvs_entry_info_t info;
    nvs_entry_info(it, &info);
    printf("key '%s', type '%d'", info.key, info.type);
    it = nvs_entry_next(it);
}
```

现在，迭代 NVS 分区的编程模式已更新为:

```c
nvs_iterator_t it = nullptr;
esp_err_t res = nvs_entry_find(<nvs_partition_name>, <namespace>, NVS_TYPE_ANY, &it);
while (res == ESP_OK) {
    nvs_entry_info_t info;
    nvs_entry_info(it, &info); // Can omit error check if parameters are
guaranteed to be non-NULL
    printf("key '%s', type '%d'", info.key, info.type);
    res = nvs_entry_next(&it);
}
```

迭代器有效性
请注意，由于函数签名的改动，如果存在参数错误，则可能从 nvs_entry_find() 获得无效迭代器。因此，请务必在使用 nvs_entry_find() 之前将迭代器初始化为 NULL，以免在调用 nvs_release_iterator() 之前进行复杂的错误检查。上述编程模式便是一个很好的例子。

删除 SD SPI 弃用的 API 结构体 sdspi_slot_config_t 和函数 sdspi_host_init_slot() 现已删除，并由结构体 sdspi_device_config_t 和函数 sdspi_host_init_device() 替代。
ROM SPI flash 在 v5.0 之前的版本中，ROM SPI flash 函数一般通过 esp32**/rom/spi_flash.h 得以体现。因此，为支持不同 ESP 芯片而编写的代码可能会充满不同目标的 ROM 头文件。此外，并非所有 API 都可以在全部的 ESP 芯片上使用。

现在，常用 API 被提取至 esp_rom_spiflash.h。尽管这不能算作重大变更，我们强烈建议您仅使用此头文件中的函数（即以 esp_rom_spiflash 为前缀并包含在 esp_rom_spiflash.h 中），以获得不同 ESP 芯片之间更佳的交叉兼容性。

为了提高 ROM SPI flash API 的可读性，以下函数也被重命名:

- esp_rom_spiflash_lock() 更名为 esp_rom_spiflash_set_bp()
- esp_rom_spiflash_unlock() 更名为 esp_rom_spiflash_clear_bp()

S PI flash 驱动 esp flash_speed_t enum 类型现已弃用。现在，您可以直接将实际时钟频率值传递给 flash 配置结构。下为配置 80MHz flash 频率的示例:

```
esp_flash_spi_device_config_t dev_cfg = {
    // Other members
    .freq_mhz = 80,
    // Other members
};
```

旧版 SPI flash 驱动 为了使 SPI flash 驱动更为稳定，v5.0 已经删除旧版 SPI flash 驱动。旧版 SPI flash 驱动程序是指自 v3.0 以来的默认 SPI flash 驱动程序，以及自 v4.0 以来启用配置选项 CONFIG_SPI_FLASH_USE_LEGACY_IMPL 的 SPI flash 驱动。从 v5.0 开始，我们将不再支持旧版 SPI flash 驱动程序。因此，旧版驱动 API 和 CONFIG_SPI_FLASH_USE_LEGACY_IMPL 配置选项均被删除，替换为新 SPI flash 驱动的 API。

<table>
<thead>
<tr>
<th>删除项目</th>
<th>替代项目</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi_flash_erase_sector()</td>
<td>esp_flash_erase_region()</td>
</tr>
<tr>
<td>spi_flash_erase_range()</td>
<td>esp_flash_erase_region()</td>
</tr>
<tr>
<td>spi_flash_write()</td>
<td>esp_flash_write()</td>
</tr>
<tr>
<td>spi_flash_read()</td>
<td>esp_flash_read()</td>
</tr>
<tr>
<td>spi_flash_write_encrypted()</td>
<td>esp_flash_write_encrypted()</td>
</tr>
<tr>
<td>spi_flash_read_encrypted()</td>
<td>esp_flash_read_encrypted()</td>
</tr>
</tbody>
</table>

备注：带有前缀 esp_flash 的新函数接受额外的 esp_flash_t* 参数。您可以直接将其设置为 NULL，从而使函数运行主 flash(esp_flash_default_chip)。

由于系统函数不再是公共函数，esp_spi_flash.h 头文件已停止使用。若要使用 flash 映射 API，请使用 spi_flash_mmap.h。

系统

跨核执行 跨核执行 (Inter-Processor Call, IPC) 不再是一个独立组件，现已被包含至 esp_system。因此，若项目的 CMakeLists.txt 文件中出现 PRIVQUIRES esp_ipc 或 REQUIRES esp_ipc，应删除这些选项，因为项目中已默认包含 esp_system 组件。
ESP 时钟  ESP 时钟 API（即以 esp_clk 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "ESP32/clk.h" 和 #include "esp_clk.h" 已被移除。如仍需使用 ESP 时钟 API（并不推荐），请使用 #include "esp_private/esp_clk.h" 来包含。

注意：私有 API 不属于稳定的 API，不会遵循 ESP-IDF 的版本演进规则，因此不推荐用户在应用中使用。

缓存错误中断 缓存错误中断 API（即以 esp_cache_err 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "ESP32/cache_err_int.h" 已被移除。如仍需使用缓存错误中断 API（并不推荐），请使用 #include "esp_private/cache_err_int.h" 来包含。

bootloader_support

- 函数 bootloader_common_get_reset_reason() 已被移除。请使用 ROM 组件中的函数 esp_rom_get_reset_reason()。
- 函数 esp_secure_boot_verify_sbv2_signature_block() 和 esp_secure_boot_verify_rsa_signature_block() 已被移除，无新的替换函数。不推荐用户直接使用以上函数。如需使用，请在 GitHub 上对该功能提交请求，并解释您需要此函数的原因。

断电 断电 API（即以 esp_brownout 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "brownout.h" 已被移除。如仍需使用断电 API（并不推荐），请使用 #include "esp_private/brownout.h" 来包含。

Trax Trax API（即以 trax_ 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "trax.h" 已被移除。如仍需使用 Trax API（并不推荐），请使用 #include "esp_private/trax.h" 来包含。

ROM components/esp32/rom/ 路径下存放的已弃用的 ROM 相关头文件已被移除（原包含路径为rom/*.h）。请使用新的特定目标的路径 components/esp_rom/include/ESP32/" (新的包含路径为"ESP32/rom/*.h")。

esp_hw_support

- 头文件 soc/cpu.h 及弃用的 CPU util 函数都已被移除。请包含 esp_cpu.h 来代替相同功能的函数。
- 头文件 hal/cpu_ll.h, hal/cpu_hal.h, hal/soc_ll.h, hal/soc_hal.h 和 interrupt_controller_hal.h 的 CPU API 函数已弃用。请包含 esp_cpu.h 来代替相同功能的函数。
- 头文件 compare_set.h 已被移除。请使用 esp_cpu.h 中提供的 esp_cpu_compare_and_set() 函数来代替。
- esp_cpu_get_ccount()、esp_cpu_set_ccount() 和 esp_cpu_in_ocd_debug_mode() 已从 esp_cpu.h 中移除。请分别使用 esp_cpu_get_cycle_count()、esp_cpu_set_cycle_count() 和 esp_cpu_dbgr_is_attached() 代替。
- 头文件 esp_intr.h 已被移除。请包含 esp_intr_alloc.h 以分配和操作中断。
- Panic API（即以 esp_panic 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "esp_panic.h" 已被移除。如仍需使用 Panic API（并不推荐），请使用 #include "esp_private/panic_reason.h" 来包含。此外，请包含 esp_debug_helpers.h 以使用与调试有关的任意辅助函数，如打印回溯。
- 头文件 soc/log.h 现更名为 esp_hw_log.h，并已更新为私有。建议用户使用 esp_log.h 头文件下的日志 API。
- 包含头文件 spinlock.h, clk_ctrl_os.h 和 rtc_wdt.h 时不应使用 soc 前缀，如 #include "spinlock.h"。
Chapter 5. 移植指南

- esp_chip_info() 命令返回芯片版本，格式为 \( \text{major} \times \text{revision} + \text{minor} \)。因此，为适应新格式，esp_chip_info_t 结构体中的 revision 被扩展为 uint16_t。

**PSRAM**

- 针对特定目标的头文件 spiram.h 及头文件 esp_spiram.h 已被移除，创建新组件 esp_psram。对于与 PSRAM 或 SPIRAM 相关的函数，请包含 esp_psram.h，并在 CMakeLists.txt 项目文件中将 esp_psram 设置为必需组件。
- esp_spiram_get_chip_size 和 esp_spiram_get_size 已被移除，请使用 esp_psram_get_size。

**eFuse**

- 函数 esp_secure_boot_read_key_digests() 的参数类型从 ets_secure_boot_key_digests_t* 更新为 esp_secure_boot_key_digests_t*。新类型与旧类型相同，仅移除了 allow_key_revoke 标志。在当前代码中，后者总是被设置为 true，并未提供额外信息。
- 针对 eFuse 晶圆增加主要修订版和次要修订版本。API esp_efuse_get_chip_ver() 与新修订不兼容，因此已被移除。请使用 API efuse_hal_get_major_chip_version()、efuse_hal_get_minor_chip_version() 或 efuse_hal_chip_revision() 来代替原有 API。

**esp_common** EXT_RAM_ATTR 已被弃用。请使用新的宏 EXT_RAM_BSS_ATTR 以将 bss 放在 PSRAM 上。

**esp_system**

- 头文件 esp_random.h, esp_mac.h 和 esp_chip_info.h 以往都通过头文件 esp_system.h 间接包含，更新后必须直接包含。已移除从 esp_system.h 中的间接包含功能。
- 回溯解析器 API（即以 esp_eh_frame_ 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "eh_frame_parser.h" 已被移除。如仍需使用回溯解析器 API（并不推荐），请使用 #include "esp_private/eh_frame_parser.h" 来包含。
- 中断看门狗定时器 API（即以 esp_int_wdt_ 为前缀的函数、类型或宏）已被更新为私有 API。因此，原先的包含路径 #include "esp_int_wdt.h" 已被移除。如仍需使用中断看门狗定时器 API（并不推荐），请使用 #include "esp_private/esp_int_wdt.h" 来包含。

**SOC 依赖性**

- Doxygen 中列出的公共 API 头文件中不会显示不稳定和非必要的 SOC 头文件，如 soc/soc.h 和 soc/rtc.h。这意味着，如果用户仍然需要这些“缺失”的头文件，就必须在代码中明确包含这些文件。
- Kconfig 选项 LEGACYINCLUDE_COMMON_HEADERS 也已被移除。
- 头文件 soc/soc_memory_types.h 已被弃用。请使用 esp_memory_utils.h。包含 soc/soc_memory_types.h 将触发构建警告，如 soc_memory_types.h is deprecated, please migrate to esp_memory_utils.h。

**应用跟踪** 其中一个时间截源已从定时器组驱动改为新的 GPTimer。Kconfig 选项已重新命名，例如 APPTRACE_SV_TS_SOURCE_TIMER00 已更改为 APPTRACE_SV_TS_SOURCE_GPTIMER。用户已无需选择组和定时器 ID。

**esp_timer** 基于 FRC2 的 esp_timer 过去可用于 ESP32，现在已被移除，更新后仅可使用更简单有效的 LAC 定时器。
ESP 镜像  ESP 镜像中关于 SPI 速度的枚举成员已重新更名。
- ESP_IMAGE_SPI_SPEED_80M 已被重新命名为 ESP_IMAGE_SPI_SPEED_DIV_1。
- ESP_IMAGE_SPI_SPEED_40M 已被重新命名为 ESP_IMAGE_SPI_SPEED_DIV_2。
- ESP_IMAGE_SPI_SPEED_26M 已被重新命名为 ESP_IMAGE_SPI_SPEED_DIV_3。
- ESP_IMAGE_SPI_SPEED_20M 已被重新命名为 ESP_IMAGE_SPI_SPEED_DIV_4。

任务看门狗定时器
- API esp_task_wdt_init() 更新后有如下变化：
  - 以结构体的形式传递配置。
  - 可将该函数配置为订阅空闲任务。
- 原先的配置选项 CONFIG_ESP_TASK_WDT 被重新命名为 CONFIG_ESP_TASK_WDT_INIT 并引入了一个新选项 CONFIG_ESP_TASK_WDT_EN。

FreeRTOS

遗留 API 及数据类型 在以往版本中, ESP-IDF 默认设置 configENABLE_BACKWARD_COMPATIBILITY 选项，因此可使用 FreeRTOS v8.0.0 之前的函数名称和数据类型。该选项现已默认禁用。因此默认情况下不再支持以往的 FreeRTOS 名称或类型。用户可以选择以下一种解决方式：
- 更新代码，删除以往的 FreeRTOS 名称或类型。
- 启用 CONFIG_FREERTOS_ENABLE_BACKWARD_COMPATIBILITY 以显式调用这些名称或类型。

任务快照 头文件 task_snapshot.h 已从 freertos/task.h 中移除。如需使用任务快照 API，请包含 freertos/task_snapshot.h。
函数 vTaskGetSnapshot() 现返回 BaseType_t，成功时返回值为 pdTRUE，失败则返回 pdFALSE。

FreeRTOS 断言 在以往版本中，FreeRTOS 断言通过 FREERTOS_ASSERT kconfig 选项独立配置，不同于系统的其他部分。该选项已被移除，现在需要通过 COMPILER_OPTIMIZATION_ASSERTION_LEVEL 来完成配置。

FreeRTOS 移植相关的宏 已移除以保证弃用 API 向后兼容性的 portmacro_deprecated.h 文件。建议使用下列函数来代替弃用 API。
- portENTER_CRITICAL_NESTED() 已被移除，请使用 portSET_INTERRUPT_MASK_FROM_ISR() 宏。
- portEXIT_CRITICAL_NESTED() 已被移除，请使用 portCLEAR_INTERRUPT_MASK_FROM_ISR() 宏。
- vPortCPUInitializeMutex() 已被移除，请使用 spinlock_initialize() 函数。
- vPortCPUAcquireMutex() 已被移除，请使用 spinlock_acquire() 函数。
- vPortCPUAcquireMutexTimeout() 已被移除，请使用 spinlock_acquire() 函数。
- vPortCPUReleaseMutex() 已被移除，请使用 spinlock_release() 函数。

应用程序更新
- 函数 esp_ota_get_app_description() 和 esp_ota_get_app_elf_sha256() 已被弃用，请分别使用 esp_app_get_description() 和 esp_app_get_elf_sha256() 函数来代替。这些函数已被移至新组件 esp_app_format。请参考头文件 esp_app_desc.h。
引邬加載程序支持

- esp_app_desc_t 结构体此前在 esp_app_format.h 中声明，现在在 esp_app_desc.h 中声明。
- 函数 bootloader_common_get_partition_description() 已更新为私有函数，请使用代替函数 esp_ota_get_partition_description()。注意，此函数的第一个参数为 esp_partition_t，而非 esp_partition_pos_t。

芯片版本 在应用程序开始加载时，引导加载程序会检查芯片版本。只有当版本为 >= CONFIG_ESP32_REV_MIN 和 <= CONFIG_ESP32_REV_MAX 全时，应用程序才能成功加载。

在 OTA 升级时，会检查应用程序头部中的版本需求和芯片版本是否符合条件。只有当版本为 >= CONFIG_ESP32_REV_MIN 和 <= CONFIG_ESP32_REV_MAX 全时，应用程序才能成功更新。

工具

IDF 监视器 IDF 监视器在波特率方面的改动如下：

- 目前，IDF 监视器需遵循自定义的控制台波特率 (CONFIG_ESP_CONSOLE_UART_BAUDRATE)，而非 115200。
- ESP-IDF v5.0 不再支持通过 menuconfig 自定义波特率。
- 支持通过设置环境变量或在命令行中使用 idf.py monitor -b <baud> 命令自定义波特率。
- 注意，为了与全局波特率 idf.py -b <baud> 保持一致，波特率参数已从 -B 改名为 -b。请运行 idf.py monitor --help 获取更多信息。

废弃指令 ESP-IDF v5.0 已将 idf.py 子命令和 cmake 目标名中的下划线 (_) 统一为连字符 (-)。使用废弃的子命令及目标名将会触发警告，建议使用更新后的版本。具体改动如下：

表 1: 废弃子命令及目标名

<table>
<thead>
<tr>
<th>废弃名</th>
<th>现用名</th>
</tr>
</thead>
<tbody>
<tr>
<td>efuse_common_table</td>
<td>efuse-common-table</td>
</tr>
<tr>
<td>efuse_custom_table</td>
<td>efuse-custom-table</td>
</tr>
<tr>
<td>erase_flash</td>
<td>erase-flash</td>
</tr>
<tr>
<td>partition_table</td>
<td>partition-table</td>
</tr>
<tr>
<td>partition_table-flash</td>
<td>partition-table-flash</td>
</tr>
<tr>
<td>post_debug</td>
<td>post-debug</td>
</tr>
<tr>
<td>show_efuse_table</td>
<td>show-efuse-table</td>
</tr>
<tr>
<td>erase_otadata</td>
<td>erase-otadata</td>
</tr>
<tr>
<td>read_otadata</td>
<td>read-otadata</td>
</tr>
</tbody>
</table>

Esptool CONFIG_ESPTOOLPY_FLASHSIZE_DETECT 选项已重命名为 CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE，且默认禁用。迁移到 ESP-IDF v5.0 的新项目和现有项目必须设置 CONFIG_ESPTOOLPY_FLASHSIZE。若因编译时 flash 大小未知而无法设置，可启用 CONFIG_ESPTOOLPY_HEADER_FLASHSIZE_UPDATE。但需注意的是，启用该项后，为在烧录期间使用 flash 大小更新二进制时不会导致摘要无效，映像后将不再附加 SHA256 摘要。

Windows 环境 基于 MSYS/MinGW 的 Windows 环境支持已在 ESP-IDF v4.0 中弃用，v5.0 则完全移除了该项服务。请使用 ESP-IDF 工具安装器设置 Windows 兼容环境。目前支持 Windows 命令行、Power Shell 和基于 Eclipse IDE 的图形用户界面等选项。此外，还可以使用支持的软件 <https://github.com/espressif/vscode-esp-idf-extension>，设置基于 VSCode 的环境。

Espressif Systems 2481 Release v5.1-dev-2066-g7869f4e151
5.1.2 从 5.0 迁移到 5.1

外设
Chapter 6

Libraries and Frameworks

6.1 Cloud Frameworks

ESP32 supports multiple cloud frameworks using agents built on top of ESP-IDF. Here are the pointers to various supported cloud frameworks’ agents and examples:

6.1.1 ESP RainMaker

ESP RainMaker is a complete solution for accelerated AIoT development. ESP RainMaker on GitHub.

6.1.2 AWS IoT

https://github.com/espressif/esp-aws-iot is an open source repository for ESP32 based on Amazon Web Services’ aws-iot-device-sdk-embedded-C.

6.1.3 Azure IoT

https://github.com/espressif/esp-azure is an open source repository for ESP32 based on Microsoft Azure’s azure-iot-sdk-c SDK.

6.1.4 Google IoT Core

https://github.com/espressif/esp-google-iot is an open source repository for ESP32 based on Google’s iot-device-sdk-embedded-c SDK.

6.1.5 Aliyun IoT

https://github.com/espressif/esp-aliyun is an open source repository for ESP32 based on Aliyun’s iotkit-embedded SDK.

6.1.6 Joylink IoT

https://github.com/espressif/esp-joylink is an open source repository for ESP32 based on Joylink’s joylink_dev_sdk SDK.
6.1.7 Tencent IoT

https://github.com/espressif/esp-welink is an open source repository for ESP32 based on Tencent’s welink SDK.

6.1.8 Tencent Yun IoT

https://github.com/espressif/esp-qcloud is an open source repository for ESP32 based on Tencent Yun’s qcloud-iot-sdk-embedded-c SDK.

6.1.9 Baidu IoT

https://github.com/espressif/esp-baidu-iot is an open source repository for ESP32 based on Baidu’s iot-sdk-c SDK.

6.2 Espressif’s Frameworks

Here you will find a collection of the official Espressif libraries and frameworks.

6.2.1 Espressif Audio Development Framework

The ESP-ADF is a comprehensive framework for audio applications including:

• CODEC’s HAL
• Music Players and Recorders
• Audio Processing
• Bluetooth Speakers
• Internet Radios
• Hands-free devices
• Speech Recognition

This framework is available at GitHub: ESP-ADF.

6.2.2 ESP-CSI

ESP-CSI is an experimental implementation that uses the Wi-Fi Channel State Information to detect the presence of a human body.

See ESP-CSI project for more information about it.

6.2.3 Espressif DSP Library

The library provides algorithms optimized specifically for digital signal processing applications. This library supports:

• Matrix multiplication
• Dot product
• FFT (Fast Fourier Transform)
• IIR (Infinite Impulse Response)
• FIR (Finite Impulse Response)
• Vector math operations

This library is available here: ESP-DSP library.
6.2.4 ESP-WIFI-MESH Development Framework

This framework is based on the ESP-WIFI-MESH protocol with the following features:

- Fast network configuration
- Stable upgrade
- Efficient debugging
- LAN control
- Various application demos

ESP-MDF.

6.2.5 ESP-WHO

The ESP-WHO is a face detection and recognition framework using the ESP32 and camera. To know more about the project, see ESP-WHO on GitHub.

6.2.6 ESP RainMaker

ESP RainMaker is a complete solution for accelerated AIoT development. Using ESP RainMaker, you can create AIoT devices from the firmware to the integration with voice-assistant, phone apps and cloud backend.

ESP RainMaker on GitHub.

6.2.7 ESP-IoT-Solution

ESP-IoT-Solution contains commonly used device drivers and code frameworks when developing IoT systems. The device drivers and code frameworks within the ESP-IoT-Solution are organized as separate components, allowing them to be easily integrated into an ESP-IDF project.

ESP-IoT-Solution includes:

- Device drivers for sensors, display, audio, GUI, input, actuators, etc.
- Framework and documentation for low power, security, storage, etc.
- Guide for Espressif open source solutions from practical application point.

ESP-IoT-Solution on GitHub.

6.2.8 ESP-Protocols

ESP-Protocols repository contains collection of protocol components for ESP-IDF. The code within the ESP-Protocols is organized into separate components, allowing them to be easily integrated into an ESP-IDF project. In addition to that, each component is available in IDF Component Registry.

ESP-Protocols components:

- `esp_modem` enables connectivity with GSM/LTE modems using AT commands or PPP protocol, see the `esp_modem documentation`.
- `mdns` (mDNS) is a multicast UDP service that is used to provide local network service and host discovery, see the `mdns documentation`.
- `esp_websocket_client` is a managed component for `esp-idf` that contains implementation of [WebSocket protocol client](https://datatracker.ietf.org/doc/html/rfc6455) for ESP32, see the `esp_websocket_client documentation`.
- `asio` is a cross-platform C++ library, see https://think-async.com/Asio/. It provides a consistent asynchronous model using a modern C++ approach., see the `asio documentation`. 
6.2.9 ESP-BSP

ESP-BSP repository contains Board Support Packages (BSPs) for various Espressif’s and 3rd party development boards. BSPs are useful for quick start on a supported board. Usually they contain pinout definition and helper functions, that will initialize peripherals for the specific board. Additionally, the BSP would contain drivers for external chips populated on the development board, such as sensors, displays, audio codecs etc.

6.2.10 ESP-IDF-CXX

ESP-IDF-CXX contains C++ wrappers for part of ESP-IDF. The focus is on ease of use, safety, automatic resource management and shifting checks to compile time instead of failing at run time. There are C++ classes for ESP-Timer, I2C, SPI, GPIO and other peripherals or features of ESP-IDF. ESP-IDF-CXX is available as a component from the component registry. Please check the project’s README.md for more information.
Chapter 7

Contributions Guide

We welcome contributions to the esp-idf project!

7.1 How to Contribute

Contributions to esp-idf - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via Github Pull Requests.

7.2 Before Contributing

Before sending us a Pull Request, please consider this list of points:

- Is the contribution entirely your own work, or already licensed under an Apache License 2.0 compatible Open Source License? If not then we unfortunately cannot accept it. Please check the Copyright Header Guide for additional information.
- Does any new code conform to the esp-idf Style Guide?
- Have you installed the pre-commit hook for esp-idf project?
- Does the code documentation follow requirements in Documenting Code?
- Is the code adequately commented for people to understand how it is structured?
- Is there documentation or examples that go with code contributions? There are additional suggestions for writing good examples in examples readme.
- Are comments and documentation written in clear English, with no spelling or grammar errors?
- Example contributions are also welcome. Please check the 创建示例项目 guide for these.
- If the contribution contains multiple commits, are they grouped together into logical changes (one major change per pull request)? Are any commits with names like “fixed typo” squashed into previous commits?
- If you’re unsure about any of these points, please open the Pull Request anyhow and then ask us for feedback.

7.3 Pull Request Process

After you open the Pull Request, there will probably be some discussion in the comments field of the request itself. Once the Pull Request is ready to merge, it will first be merged into our internal git system for in-house automated testing.

If this process passes, it will be merged onto the public github repository.
Chapter 7. Contributions Guide

7.4 Legal Part

Before a contribution can be accepted, you will need to sign our Contributor Agreement. You will be prompted for this automatically as part of the Pull Request process.

7.5 Related Documents

7.5.1 Espressif IoT Development Framework Style Guide

About This Guide

Purpose of this style guide is to encourage use of common coding practices within the ESP-IDF. Style guide is a set of rules which are aimed to help create readable, maintainable, and robust code. By writing code which looks the same way across the code base we help others read and comprehend the code. By using same conventions for spaces and newlines we reduce chances that future changes will produce huge unreadable diffs. By following common patterns for module structure and by using language features consistently we help others understand code behavior.

We try to keep rules simple enough, which means that they can not cover all potential cases. In some cases one has to bend these simple rules to achieve readability, maintainability, or robustness.

When doing modifications to third-party code used in ESP-IDF, follow the way that particular project is written. That will help propose useful changes for merging into upstream project.

C Code Formatting

Naming

- Any variable or function which is only used in a single source file should be declared static.
- Public names (non-static variables and functions) should be namedpaced with a per-component or per-unit prefix, to avoid naming collisions. i.e esp_vfs_register() or esp_console_run(). Starting the prefix with esp_ for Espressif-specific names is optional, but should be consistent with any other names in the same component.
- Static variables should be prefixed with s_ for easy identification. For example, static bool s_invert.
- Avoid unnecessary abbreviations (i.e shortening data to dat), unless the resulting name would otherwise be very long.

Indentation

Use 4 spaces for each indentation level. Don’t use tabs for indentation. Configure the editor to emit 4 spaces each time you press tab key.

Vertical Space

Place one empty line between functions. Don’t begin or end a function with an empty line.
The maximum line length is 120 characters as long as it doesn’t seriously affect the readability.

**Horizontal Space** Always add single space after conditional and loop keywords:

```c
if (condition) {  // correct
  // ...
}
switch (n) {      // correct
  case 0:
    // ...
}
for (int i = 0; i < CONST; ++i) {  // INCORRECT
  // ...
}
```

Add single space around binary operators. No space is necessary for unary operators. It is okay to drop space around multiply and divide operators:

```c
const int y = y0 + (x - x0) * (y1 - y0) / (x1 - x0);    // correct
const int y = y0 + (x - x0)*(y1 - y0)/(x1 - x0);      // also okay
int y_cur = -y;
  ++y_cur;
const int y = y0+(x-x0)*(y1-y0)/(x1-x0);              // INCORRECT
```

No space is necessary around . and -> operators.
Sometimes adding horizontal space within a line can help make code more readable. For example, you can add space to align function arguments:

```c
esp_rom_gpio_connect_in_signal(PIN_CAM_D6, I2S0I_DATA_IN14_IDX, false);
esp_rom_gpio_connect_in_signal(PIN_CAM_D7, I2S0I_DATA_IN15_IDX, false);
esp_rom_gpio_connect_in_signal(PIN_CAM_HREF, I2S0I_H_ENABLE_IDX, false);
esp_rom_gpio_connect_in_signal(PIN_CAM_PCLK, I2S0I_DATA_IN15_IDX, false);
```

Note however that if someone goes to add new line with a longer identifier as first argument (e.g. PIN_CAM_VSYNC), it will not fit. So other lines would have to be realigned, adding meaningless changes to the commit.

Therefore, use horizontal alignment sparingly, especially if you expect new lines to be added to the list later.

Never use TAB characters for horizontal alignment.
Never add trailing whitespace at the end of the line.

**Braces**

- Function definition should have a brace on a separate line:

  ```c
  // This is correct:
  void function(int arg)
  {
  }
  ```
• Within a function, place opening brace on the same line with conditional and loop statements:

```c
if (condition) {
    do_one();
} else if (other_condition) {
    do_two();
}
```

**Comments**  Use `//` for single line comments. For multi-line comments it is okay to use either `//` on each line or a `/* */` block.

Although not directly related to formatting, here are a few notes about using comments effectively.

• Don’t use single comments to disable some functionality:

```c
void init_something()
{
    setup_dma();
    // load_resources();  // WHY is this thing commented, asks...
    --the reader?
    start_timer();
}
```

• If some code is no longer required, remove it completely. If you need it you can always look it up in git history of this file. If you disable some call because of temporary reasons, with an intention to restore it in the future, add explanation on the adjacent line:

```c
void init_something()
{
    setup_dma();
    // TODO: we should load resources here, but loader is not fully integrated...
    --yet.
    // load_resources();
    start_timer();
}
```

• Same goes for `#if 0 ... #endif` blocks. Remove code block completely if it is not used. Otherwise, add comment explaining why the block is disabled. Don’t use `#if 0 ... #endif` or comments to store code snippets which you may need in the future.

• Don’t add trivial comments about authorship and change date. You can always look up who modified any given line using git. E.g. this comment adds clutter to the code without adding any useful information:

```c
void init_something()
{
    setup_dma();
    // XXX add 2016-09-01
    init_dma_list();
    fill_dma_item(0);
    // end XXX add
    start_timer();
}
```

**Line Endings**  Commits should only contain files with LF (Unix style) endings.
Windows users can configure git to check out CRLF (Windows style) endings locally and commit LF endings by setting the `core.autocrlf` setting. Github has a document about setting this option <github-line-endings>.

If you accidentally have some commits in your branch that add LF endings, you can convert them to Unix by running this command in an MSYS2 or Unix terminal (change directory to the IDF working directory and check the correct branch is currently checked out, beforehand):

```
git rebase --exec 'git diff-tree --no-commit-id --name-only -r HEAD | xargs --dos2unix & & git commit -a --amend --no-edit --allow-empty' master
```

(Note that this line rebases on master, change the branch name at the end to rebase on another branch.)

For updating a single commit, it’s possible to run `dos2unix FILENAME` and then run `git commit --amend`

**Formatting Your Code** You can use `astyle` program to format your code according to the above recommendations.

If you are writing a file from scratch, or doing a complete rewrite, feel free to re-format the entire file. If you are changing a small portion of file, don’t re-format the code you didn’t change. This will help others when they review your changes.

To re-format a file, run:

```
tools/format.sh components/my_component/file.c
```

**Type Definitions** Should be snake_case, ending with _t suffix:

```c
typedef int signed_32_bit_t;
```

**Enum** Enums should be defined through the `typedef` and be namespaced:

```c
typedef enum
{
    MODULE_FOO_ONE,
    MODULE_FOO_TWO,
    MODULE_FOO_THREE
} module_foo_t;
```

**Assertions** The standard C `assert()` function, defined in `assert.h` should be used to check conditions that should be true in source code. In the default configuration, an assert condition that returns false or 0 will call `abort()` and trigger a **Fatal Error**.

`assert()` should only be used to detect unrecoverable errors due to a serious internal logic bug or corruption, where it’s not possible for the program to continue. For recoverable errors, including errors that are possible due to invalid external input, an error value should be returned.

**备注:** When asserting a value of type `esp_err_t` is equal to `ESP_OK`, use the `ESP_ERROR_CHECK` macro instead of an `assert()`.

It’s possible to configure ESP-IDF projects with assertions disabled (see `CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL`). Therefore, functions called in an `assert()` statement should not have side-effects.

It’s also necessary to use particular techniques to avoid “variable set but not used” warnings when assertions are disabled, due to code patterns such as:
```cpp
int res = do_something();
assert(res == 0);
```

Once the `assert` is optimized out, the `res` value is unused and the compiler will warn about this. However the function `do_something()` must still be called, even if assertions are disabled.

When the variable is declared and initialized in a single statement, a good strategy is to cast it to `void` on a new line. The compiler will not produce a warning, and the variable can still be optimized out of the final binary:

```cpp
int res = do_something();
assert(res == 0);
(void) res;
```

If the variable is declared separately, for example if it is used for multiple assertions, then it can be declared with the GCC attribute `__attribute__((unused))`. The compiler will not produce any unused variable warnings, but the variable can still be optimized out:

```cpp
int res __attribute__((unused));
res = do_something();
assert(res == 0);
res = do_something_else();
assert(res != 0);
```

### Header file guards

All public facing header files should have preprocessor guards. A pragma is preferred:

```cpp
#pragma once
```

over the following pattern:

```
ifndef FILE_NAME_H
#define FILE_NAME_H
...
#endif // FILE_NAME_H
```

In addition to guard macros, all C header files should have `extern "C"` guards to allow the header to be used from C++ code. Note that the following order should be used: `pragma once`, then any `#include` statements, then `extern "C"` guards:

```cpp
#pragma once
#include <stdint.h>
#ifndef __cplusplus
extern "Cplusplus"
extern "C" {
#endif
/* declarations go here */
#ifndef __cplusplus
}
#endif
```

### Include statements

When writing `#include` statements, try to maintain the following order:
Chapter 7. Contributions Guide

- C standard library headers.
- Other POSIX standard headers and common extensions to them (such as sys/queue.h.)
- Common IDF headers (esp_log.h, esp_system.h, esp_timer.h, esp_sleep.h, etc.)
- Headers of other components, such as FreeRTOS.
- Public headers of the current component.
- Private headers.

Use angle brackets for C standard library headers and other POSIX headers (#include <stdio.h>.

Use double quotes for all other headers (#include "esp_log.h").

C++ Code Formatting

The same rules as for C apply. Where they are not enough, apply the following rules.

File Naming  C++ Header files have the extension .hpp. C++ source files have the extension .cpp. The latter is important for the compiler to distinguish them from normal C source files.

Naming

- **Class and struct** names shall be written in CamelCase with a capital letter as beginning. Member variables and methods shall be in snake_case.
- **Namespaces** shall be in lower snake_case.
- **Templates** are specified in the line above the function declaration.
- Interfaces in terms of Object-Oriented Programming shall be named without the suffix ...Interface. Later, this makes it easier to extract interfaces from normal classes and vice versa without making a breaking change.

Member Order in Classes  In order of precedence:

- First put the public members, then the protected, then private ones. Omit public, protected or private sections without any members.
- First put constructors/destructors, then member functions, then member variables.

For example:

```cpp
class ForExample {
public:
  // first constructors, then default constructor, then destructor
  ForExample(double example_factor_arg);
  ForExample();
  ~ForExample();

  // then remaining public methods
  set_example_factor(double example_factor_arg);

  // then public member variables
  uint32_t public_data_member;

private:
  // first private methods
  void internal_method();

  // then private member variables
  double example_factor;
};
```
Spacing

- Don’t indent inside namespaces.
- Put public, protected and private labels at the same indentation level as the corresponding class label.

Simple Example

```c++
// file spaceship.h
#ifndef SPACESHIP_H_
#define SPACESHIP_H_
#include <cstdlib>
namespace spaceships {
    class SpaceShip {
    public:
        SpaceShip(size_t crew);
        size_t get_crew_size() const;
    private:
        const size_t crew;
    };
    class SpaceShuttle : public SpaceShip {
    public:
        SpaceShuttle();
    };
    class Sojuz : public SpaceShip {
    public:
        Sojuz();
    };
    template<typename T>
    class CargoShip {
    public:
        CargoShip(const T &cargo);
    private:
        T cargo;
    };
} // namespace spaceships
#endif // SPACESHIP_H_

// file spaceship.cpp
#include "spaceship.h"
namespace spaceships {
    // Putting the curly braces in the same line for constructors is OK if it only initializes
    // values in the initializer list
    SpaceShip::SpaceShip(size_t crew) : crew(crew) { }
    size_t SpaceShip::get_crew_size() const {
        return crew;
    }
} // namespace spaceships
```
CMake Code Style

- Indent with four spaces.
- Maximum line length 120 characters. When splitting lines, try to focus on readability where possible (for example, by pairing up keyword/argument pairs on individual lines).
- Don’t put anything in the optional parentheses after endforeach(), endif(), etc.
- Use lowercase (with_underscores) for command, function, and macro names.
- For locally scoped variables, use lowercase (with_underscores).
- For globally scoped variables, use uppercase (WITH_UNDERSCORES).
- Otherwise follow the defaults of the cmake-lint project.

Configuring the Code Style for a Project Using EditorConfig

EditorConfig helps developers define and maintain consistent coding styles between different editors and IDEs. The EditorConfig project consists of a file format for defining coding styles and a collection of text editor plugins that enable editors to read the file format and adhere to defined styles. EditorConfig files are easily readable and they work nicely with version control systems.

For more information, see EditorConfig Website.

Third Party Component Code Styles

ESP-IDF integrates a number of third party components where these components may have differing code styles.

FreeRTOS The code style adopted by FreeRTOS is described in the FreeRTOS style guide. Formatting of FreeRTOS source code is automated using Uncrustify, thus a copy of the FreeRTOS code style’s Uncrustify configuration (uncrustify.cfg) is stored within ESP-IDF FreeRTOS component.

If a FreeRTOS source file is modified, the updated file can be formatted again by following the steps below:

1. Ensure that Uncrustify (v0.69.0) is installed on your system
2. Run the following command on the update FreeRTOS source file (where source.c is the path to the source file that requires formatting).

   uncrustify -c "$IDF_PATH/components/freertos/FreeRTOS-Kernel/uncrustify.cfg" --replace source.c --no-backup

Documenting Code

Please see the guide here: Documenting Code.
Chapter 7. Contributions Guide

Structure
To be written.

Language Features
To be written.

7.5.2 Install pre-commit Hook for ESP-IDF Project

Required Dependency
Python 3.7.* or above. This is our recommended python version for IDF developers.
If you still have python versions not compatible, update your python versions before installing the pre-commit hook.

Install pre-commit

Run `pip install pre-commit`

Install pre-commit hook

1. Go to the IDF Project Directory
2. Run `pre-commit install --allow-missing-config`. Install hook by this approach will let you commit successfully even in branches without the `.pre-commit-config.yaml`
3. pre-commit hook will run automatically when you’re running `git commit` command

Uninstall pre-commit

Run `pre-commit uninstall`

What’s More?

For detailed usage, please refer to the documentation of pre-commit.

Common Problems For Windows Users

`/usr/bin/env: python: Permission denied`. If you’re in Git Bash, please check the python executable location by run `which python`
If the executable is under `~/.AppData/Local/Microsoft/WindowsApps/`, then it’s a link to Windows AppStore, not a real one.
Please install python manually and update this in your PATH environment variable.

Your `%USERPROFILE%` contains non-ASCII characters
pre-commit may fail when initializing an environment for a particular hook when the path of pre-commit’s cache contains non-ASCII characters. The solution is to set `PRE_COMMIT_HOME` to a path containing only standard characters before running pre-commit.

- **CMD:** `set PRE_COMMIT_HOME=C:\somepath\pre-commit`
- **PowerShell:** `$Env:PRE_COMMIT_HOME = "C:\somepath\pre-commit"`
- **git bash:** `export PRE_COMMIT_HOME="/c/somepath/pre-commit"`
7.5.3 Documenting Code

The purpose of this description is to provide quick summary on documentation style used in espresif/esp-idf repository and how to add new documentation.

Introduction

When documenting code for this repository, please follow Doxygen style. You are doing it by inserting special commands, for instance @param, into standard comments blocks, for example:

```c
/**
 * @param ratio this is oxygen to air ratio
 */
```

Doxygen is phrasing the code, extracting the commands together with subsequent text, and building documentation out of it.

Typical comment block, that contains documentation of a function, looks like below.

Doxygen supports couple of formatting styles. It also gives you great flexibility on level of details to include in documentation. To get familiar with available features, please check data rich and very well organized Doxygen Manual.

Why we need it?

The ultimate goal is to ensure that all the code is consistently documented, so we can use tools like Sphinx and Breathe to aid preparation and automatic updates of API documentation when the code changes.

With these tools the above piece of code renders like below:
Chapter 7. Contributions Guide

Go for it!

When writing code for this repository, please follow guidelines below.

1. Document all building blocks of code: functions, structs, typedefs, enums, macros, etc. Provide enough information about purpose, functionality and limitations of documented items, as you would like to see them documented when reading the code by others.

2. Documentation of function should describe what this function does. If it accepts input parameters and returns some value, all of them should be explained.

3. Do not add a data type before parameter or any other characters besides spaces. All spaces and line breaks are compressed into a single space. If you like to break a line, then break it twice.

4. If function has void input or does not return any value, then skip @param or @return
5. When documenting a `define` as well as members of a `struct` or `enum`, place specific comment like below after each member.

6. To provide well formatted lists, break the line after command (like `@return` in example below).

```
* @return
  * - ESP_OK if erase operation was successful
  * - ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
  * - ESP_ERR_NVS_READ_ONLY if handle was opened as read only
  * - ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
  * - other error codes from the underlying storage driver
```

7. Overview of functionality of documented header file, or group of files that make a library, should be placed in a separate README.rst file of the same directory. If this directory contains header files for different APIs, then the file name should be apiname-readme.rst.

---

**Go one extra mile**

Here are a couple of tips on how you can make your documentation even better and more useful to the reader and writer.

When writing codes, please follow the guidelines below:

1. Add code snippets to illustrate implementation. To do so, enclose snippet using `@code{c}` and `@endcode` commands.

```
* @code{c}
* // Example of using nvs_get_i32:
* int32_t max_buffer_size = 4096; // default value
* esp_err_t err = nvs_get_i32(my_handle, "max_buffer_size", &max_buffer_size);
* assert(err == ESP_OK || err == ESP_ERR_NVS_NOT_FOUND);
* // if ESP_ERR_NVS_NOT_FOUND was returned, max_buffer_size will still
* // have its default value.
```
Chapter 7. Contributions Guide

The code snippet should be enclosed in a comment block of the function that it illustrates.

2. To highlight some important information use command `@attention` or `@note`.

```
@attention
1. This API only impact WIFI_MODE_STA or WIFI_MODE_APSTA mode
2. If the ESP32 is connected to an AP, call esp_wifi_disconnect to disconnect.
```

Above example also shows how to use a numbered list.

3. To provide common description to a group of similar functions, enclose them using `/**@{*/` and `/**}@*/` markup commands:

```
/**@{*/
/**
* @brief common description of similar functions
*/
void first_similar_function (void);
void second_similar_function (void);
/**}@*/
```

For practical example see `nvs_flash/include/nvs.h`.

4. You may want to go even further and skip some code like repetitive defines or enumerations. In such case, enclose the code within `/** @cond */` and `/** @endcond */` commands. Example of such implementation is provided in `driver/include/driver/gpio.h`.

5. Use markdown to make your documentation even more readable. You will add headers, links, tables and more.

```
```

6. Prepare one or more complete code examples together with description. Place description to a separate file `README.md` in specific folder of `examples` directory.

**Standardize Document Format**

When it comes to text, please follow guidelines below to provide well formatted Markdown (.md) or reST (.rst) documents.

1. Please ensure that one paragraph is written in one line. Don’t break lines like below. Breaking lines to enhance readability is only suitable for writing codes. To make the text easier to read, it is recommended to place an empty line to separate the paragraph.

2. Please make the line number of CN and EN documents consistent like below. The benefit of this approach is that it can save time for both writers and translators. When non-bilingual writers need to update text, they only need to update the same line in the corresponding CN or EN document. For translators, if documents are updated in English, then translators can quickly locate where to update in the corresponding CN document later. Besides, by comparing the total number of lines in EN and CN documents, you can quickly find out whether the CN version lags behind the EN version.
Recommend: one line for one paragraph like below

To realize the multiplexing of different devices from different drivers (SPI Master, SPI Flash, etc.), an SPI bus lock is applied on each SPI bus. Drivers can attach their devices onto the bus with the arbitration of the lock.

Each bus lock is initialized with a BG (background) service registered. All devices request to do transactions on the bus should wait until the BG to be successfully disabled.

- For SPI bus, the BG is the cache, the bus lock will help to disable the cache before device operations starts, and enable it again after device releasing the lock. No devices on SPI2 is allowed using ISR (it's meaningless for the task to yield to other tasks when the cache is disabled).

Don't need to break lines here

To realize the multiplexing of different devices from different drivers (SPI Master, SPI Flash, etc.), an SPI bus lock is applied on each SPI bus. Drivers can attach their devices onto the bus with the arbitration of the lock.

Each bus lock is initialized with a BG (background) service registered. All devices request to do transactions on the bus should wait until the BG to be successfully disabled.

- For SPI bus, the BG is the cache, the bus lock will help to disable the cache before device operations starts, and enable it again after device releasing the lock. No devices on SPI2 is allowed using ISR (it's meaningless for the task to yield to other tasks when the cache is disabled).

Keep the line number for EN and CN documents consistent
Building Documentation

The documentation is built with the esp-docs Python package, which is a wrapper around Sphinx.

To install it simply do:

```
pip install esp-docs
```

After a successful install then the documentation can be built from the docs folder with:

```
built-docs build
```

or for specific target and language with:

```
built-docs -t esp32 -l en build
```

For more in-depth documentation about esp-docs features please see the documentation at esp-docs.

Wrap up

We love good code that is doing cool things. We love it even better, if it is well documented, so we can quickly make it run and also do the cool things.

Go ahead, contribute your code and documentation!

Related Documents

- API Documentation Template

7.5.4 创建示例项目

每个 ESP-IDF 的示例都是一个完整的项目，其他人可以将示例复制至本地，并根据实际情况进行一定修改。请注意，示例项目主要是为了展示 ESP-IDF 的功能。

示例项目结构

- main 目录需要包含一个名为 (something)_example_main.c 的源文件，里面包含示例项目的主功能。
- 如果该示例项目的子任务比较多，请根据逻辑将其拆分为 main 目录下的多个 C 或者 C++ 源文件，并将对应的头文件也放在同一目录下。
- 如果该示例项目具有多种功能，可以考虑在项目中增加一个 components 子目录，通过库功能，将示例项目的不同功能划分为不同的组件。注意，如果该组件提供的功能相对完整，且具有一定的通用性，则应该将它们添加到 ESP-IDF 的 components 目录中，使其成为 ESP-IDF 的一部分。
- 示例项目需要包含一个 README.md 文件，建议使用示例项目 README 模板，并根据项目实际情况进行修改。
- 示例项目需要包含一个 example_test.py 文件，用于进行自动化测试。如果在 GitHub 上初次提交 Pull Request 时，可以先不包含这个脚本文件。具体细节，请见有关 Pull Request 的相关内容。

一般准则

示例代码需要遵循《乐鑫物联网开发框架风格指南》。
检查清单

提交一个新的示例项目之前，需要检查以下内容：

- 示例项目的名字（包括 README.md 中）应使用 example，而不要写“demo”、“test”等词汇。
- 每个示例项目只能有一个主要功能。如果某个示例项目有多个主要功能，请将其拆分为两个或更多示例项目。
- 每个示例项目应包含一个 README.md 文件。建议使用 示例项目 README 模板。
- 示例项目中的函数和变量的命名要遵循命名规范中的要求。对于仅在示例项目源文件中使用的非静态变量/函数，请使用 example 或其他类似的前缀。
- 示例项目中的所有代码结构良好，关键代码要有详细注释。
- 示例项目中所有不必要的代码（旧的调试日志，注释掉的代码等）都必须清除掉。
- 示例项目中使用的选项（比如网络名称，地址等）不得直接硬编码，应尽可能地使用配置项，或者定义为宏或变量。
- 配置项可见 KConfig.projbuild 文件，该文件中包含一个名为“Example Configuration”的菜单。
- 具体情况，请查看现有示例项目。
- 所有的源代码都需要在文件开头指定许可信息（表示该代码是 in the public domain CC0）和免责声明。或者，源代码也可以应用 Apache License 2.0 许可条款。请查看现有示例项目中的许可信息和免责声明，并根据实际情况进行修改。
- 任何第三方代码（无论是直接使用，还是进行了一些改进）均应保留原始代码中的许可信息，且这些代码的许可必须兼容 Apache License 2.0 协议。

7.5.5 API Documentation Template

备注：INSTRUCTIONS

1. Use this file (docs/en/api-reference/template.rst) as a template to document API.
2. Change the file name to the name of the header file that represents documented API.
3. Include respective files with descriptions from the API folder using ..include::
   - README.rst
   - example.rst
   - ...
4. Optionally provide description right in this file.
5. Once done, remove all instructions like this one and any superfluous headers.

Overview

备注：INSTRUCTIONS

1. Provide overview where and how this API may be used.
2. Where applicable include code snippets to illustrate functionality of particular functions.
3. To distinguish between sections, use the following heading levels:
   - # with overline, for parts
   - * with overline, for chapters
   - =, for sections
   - -, for subsections
   - ^, for subsubsections
   - "", for paragraphs

Application Example

备注：INSTRUCTIONS
1. Prepare one or more practical examples to demonstrate functionality of this API.
2. Each example should follow pattern of projects located in `esp-idf/examples/` folder.
3. Place example in this folder complete with `README.md` file.
4. Provide overview of demonstrated functionality in `README.md`.
5. With good overview reader should be able to understand what example does without opening the source code.
6. Depending on complexity of example, break down description of code into parts and provide overview of functionality of each part.
7. Include flow diagram and screenshots of application output if applicable.
8. Finally add in this section synopsis of each example together with link to respective folder in `esp-idf/examples/`.

**API Reference**

**INSTRUCTIONS**

1. This repository provides for automatic update of API reference documentation using code markup retrieved by Doxygen from header files.
2. Update is done on each documentation build by invoking Sphinx extension `esp_extensions/run_doxygen.py` for all header files listed in the `INPUT` statement of `docs/doxygen/Doxyfile`.
3. Each line of the `INPUT` statement (other than a comment that begins with `##`) contains a path to header file `*.h` that will be used to generate corresponding `*.inc` files:

```
## Wi-Fi - API Reference
../components/esp32/include/esp_wifi.h \
../components/esp32/include/esp_smartconfig.h \
```

4. When the headers are expanded, any macros defined by default in `sdkconfig.h` as well as any macros defined in SOC-specific `include/soc/*.caps.h` headers will be expanded. This allows the headers to include/exclude material based on the `IDF_TARGET` value.
5. The `*.inc` files contain formatted reference of API members generated automatically on each documentation build. All `*.inc` files are placed in Sphinx `_build` directory. To see directives generated for e.g. `esp_wifi.h`, run `python gen-dxd.py esp32/include/esp_wifi.h`.
6. To show contents of `*.inc` file in documentation, include it as follows:

```
.. include-build-file:: inc/esp_wifi.inc
```

For example see `docs/en/api-reference/network/esp_wifi.rst`

Below is the list of common .. doxygen...:: directives:

- Functions ... doxygenfunction:: name_of_function
- Unions ... doxygenuinion:: name_of_union
- Structures ... doxygenstruct:: name_of_structure together with :members:
- Macros ... doxygendefine:: name_of_define
- Type Definitions ... doxygentypedef:: name_of_type
- Enumerations ... doxygenenum:: name_of Enumeration

See Breathe documentation for additional information.

To provide a link to header file, use the `link custom role` directive as follows:

```
*:component_file:`path_to/header_file.h`
```

8. In any case, to generate API reference, the file `docs/doxygen/Doxyfile` should be updated with paths to `*.h` headers that are being documented.
9. When changes are committed and documentation is build, check how this section has been rendered. Correct annotations in respective header files, if required.
7.5.6 Contributor Agreement

**Individual Contributor Non-Exclusive License Agreement including the Traditional Patent License OPTION**

Thank you for your interest in contributing to Espressif IoT Development Framework (esp-idf) ("We" or "Us").

The purpose of this contributor agreement ("Agreement") is to clarify and document the rights granted by contributors to Us. To make this document effective, please follow the instructions in the Contributions Guide.

1. **DEFINITIONS**

   "You" means the Individual Copyright owner who submits a Contribution to Us. If You are an employee and submit the Contribution as part of your employment, You have had Your employer approve this Agreement or sign the Entity version of this document.

   "Contribution" means any original work of authorship (software and/or documentation) including any modifications or additions to an existing work, Submitted by You to Us, in which You own the Copyright. If You do not own the Copyright in the entire work of authorship, please contact Us at marius.vikhammer@espressif.com.

   "Copyright" means all rights protecting works of authorship owned or controlled by You, including copyright, moral and neighboring rights, as appropriate, for the full term of their existence including any extensions by You.

   "Material" means the software or documentation made available by Us to third parties. When this Agreement covers more than one software project, the Material means the software or documentation to which the Contribution was Submitted. After You Submit the Contribution, it may be included in the Material.

   "Submit" means any form of physical, electronic, or written communication sent to Us, including but not limited to electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, Us, but excluding communication that is conspicuously marked or otherwise designated in writing by You as "Not a Contribution."

   "Submission Date" means the date You Submit a Contribution to Us.

   "Documentation" means any non-software portion of a Contribution.

2. **LICENSE GRANT**

   2.1 Copyright License to Us

   Subject to the terms and conditions of this Agreement, You hereby grant to Us a worldwide, royalty-free, NON-exclusive, perpetual and irrevocable license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, under the Copyright covering the Contribution to use the Contribution by all means, including, but not limited to:

   • to publish the Contribution,
   • to modify the Contribution, to prepare derivative works based upon or containing the Contribution and to combine the Contribution with other software code,
   • to reproduce the Contribution in original or modified form,
   • to distribute, to make the Contribution available to the public, display and publicly perform the Contribution in original or modified form.

2.2 Moral Rights remain unaffected to the extent they are recognized and not waivable by applicable law. Notwithstanding, You may add your name in the header of the source code files of Your Contribution and We will respect this attribution when using Your Contribution.

3. **PATENTS**

   3.1 Patent License

   Subject to the terms and conditions of this Agreement You hereby grant to Us a worldwide, royalty-free, non-exclusive, perpetual and irrevocable license (except as stated in Section 3.2) patent license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, to make, have made, use, sell, offer for sale, import and otherwise transfer the Contribution and the Contribution in combination with the Material (and portions of such combination). This license applies to all patents owned or controlled by You, whether already acquired or hereafter acquired, that would be infringed by making, having made, using, selling, offering for sale, importing or otherwise transferring of Your Contribution(s) alone or by combination of Your Contribution(s) with the Material.

3.2 Revocation of Patent License
You reserve the right to revoke the patent license stated in section 3.1 if we make any infringement claim that is targeted at your Contribution and not asserted for a Defensive Purpose. An assertion of claims of the Patents shall be considered for a “Defensive Purpose” if the claims are asserted against an entity that has filed, maintained, threatened, or voluntarily participated in a patent infringement lawsuit against Us or any of Our licensees.

4. DISCLAIMER  THE CONTRIBUTION IS PROVIDED “AS IS”. MORE PARTICULARLY, ALL EXPRESS OR IMPLIED WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED BY YOU TO US AND BY US TO YOU. TO THE EXTENT THAT ANY SUCH WARRANTIES CANNOT BE DISCLAIMED, SUCH WARRANTY IS LIMITED IN DURATION TO THE MINIMUM PERIOD PERMITTED BY LAW.

5. Consequential Damage Waiver  TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL YOU OR US BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF ANTICIPATED SAVINGS, LOSS OF DATA, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL AND EXEMPLARY DAMAGES ARISING OUT OF THIS AGREEMENT REGARDLESS OF THE LEGAL OR EQUITABLE THEORY (CONTRACT, TORT OR OTHERWISE) UPON WHICH THE CLAIM IS BASED.

6. Approximation of Disclaimer and Damage Waiver  IF THE DISCLAIMER AND DAMAGE WAIVER MENTIONED IN SECTION 4 AND SECTION 5 CANNOT BE GIVEN LEGAL EFFECT UNDER APPLICABLE LOCAL LAW, REVIEWING COURTS SHALL APPLY LOCAL LAW THAT MOST CLOSELY APPROXIMATES AN ABSOLUTE WAIVER OF ALL CIVIL LIABILITY IN CONNECTION WITH THE CONTRIBUTION.

7. Term  7.1 This Agreement shall come into effect upon Your acceptance of the terms and conditions.

7.2 In the event of a termination of this Agreement Sections 4, 5, 6, 7 and 8 shall survive such termination and shall remain in full force thereafter. For the avoidance of doubt, Contributions that are already licensed under a free and open source license at the date of the termination shall remain in full force after the termination of this Agreement.

8. Miscellaneous  8.1 This Agreement and all disputes, claims, actions, suits or other proceedings arising out of this agreement or relating in any way to it shall be governed by the laws of People’s Republic of China excluding its private international law provisions.

8.2 This Agreement sets out the entire agreement between You and Us for Your Contributions to Us and overrides all other agreements or understandings.

8.3 If any provision of this Agreement is found void and unenforceable, such provision will be replaced to the extent possible with a provision that comes closest to the meaning of the original provision and that is enforceable. The terms and conditions set forth in this Agreement shall apply notwithstanding any failure of essential purpose of this Agreement or any limited remedy to the maximum extent possible under law.

8.4 You agree to notify Us of any facts or circumstances of which you become aware that would make this Agreement inaccurate in any respect.

You

Date: 
Name: 
Title: 
Address: 

Espressif Systems  2506 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
7.5.7 Copyright Header Guide

ESP-IDF is released under the Apache License 2.0 with some additional third-party copyrighted code released under various licenses. For further information please refer to the list of copyrights and licenses.

This page explains how the source code should be properly marked with a copyright header. ESP-IDF uses The Software Package Data Exchange (SPDX) format which is short and can be easily read by humans or processed by automated tools for copyright checks.

How to Check the Copyright Headers

Please make sure you have installed the pre-commit hooks which contain a copyright header checker as well. The checker can suggest a header if it is not able to detect a properly formatted SPDX header.

What if the Checker’s Suggestion is Incorrect?

No automated checker (no matter how good is) can replace humans. So the developer’s responsibility is to modify the offered header to be in line with the law and the license restrictions of the original code on which the work is based on. Certain licenses are not compatible between each other. Such corner cases will be covered by the following examples.

The checker can be configured with the tools/ci/check_copyright_config.yaml configuration file. Please check the options it offers and consider updating it in order to match the headers correctly.

Common Examples of Copyright Headers

The simplest case is when the code is not based on any licensed previous work, e.g. it was written completely from scratch. Such code can be decorated with the following copyright header and put under the license of ESP-IDF:

```c
/*
 * SPDX-License-Identifier: Apache-2.0
 */
```

Less restrictive parts of ESP-IDF Some parts of ESP-IDF are deliberately under less restrictive licenses in order to ease their re-use in commercial closed source projects. This is the case for ESP-IDF examples which are in Public domain or under the Creative Commons Zero Universal (CC0) license. The following header can be used in such source files:

```c
/*
 * SPDX-License-Identifier: Unlicense OR CC0-1.0
 */
```
The option allowing multiple licenses joined with the `OR` keyword from the above example can be achieved with the definition of multiple allowed licenses in the `tools/ci/check_copyright_config.yaml` configuration file. Please use this option with care and only selectively for a limited part of ESP-IDF.

**Third party licenses** Code licensed under different licenses, modified by Espressif Systems and included in ESP-IDF cannot be licensed under Apache License 2.0 even if the checker suggests it. It is advised to keep the original copyright header and add an SPDX before it.

The following example is a suitable header for a code licensed under the “GNU General Public License v2.0 or later” held by John Doe with some additional modifications done by Espressif Systems:

```plaintext
/*
 * SPDX-FileCopyrightText: 1991 John Doe
 * SPDX-License-Identifier: GPL-2.0-or-later
 * SPDX-FileContributor: 2019-2022 Espressif Systems (Shanghai) CO LTD
 */
```

The licenses can be identified and the short SPDX identifiers can be found in the official SPDX license list. Other very common licenses are the GPL-2.0-only, the BSD-3-Clause, and the BSD-2-Clause.

The configuration stored in `tools/ci/check_copyright_config.yaml` offers features useful for third party licenses:

- A different license can be defined for the files part of a third party library.
- The check for a selected set of files can be permanently disabled. Please use this option with care and only in cases when none of the other options are suitable.

### 7.5.8 ESP-IDF Tests with Pytest Guide

This documentation is a guide that introduces the following aspects:

1. The basic idea of different test types in ESP-IDF
2. How to apply the pytest framework to the test python scripts to make sure the apps are working as expected.
3. ESP-IDF CI target test process
4. Run ESP-IDF tests with pytest locally
5. Tips and tricks on pytest

**Disclaimer**

In ESP-IDF, we use the following plugins by default:

- `pytest-embedded` with default services `esp, idf`
- `pytest-rerunfailures`

All the introduced concepts and usages are based on the default behavior in ESP-IDF. Not all of them are available in vanilla pytest.

**Installation**

All dependencies could be installed by running the install script with the `--enable-pytest` argument, e.g. `$ install.sh --enable-pytest`.

**Common Issues During Installation**
No Package ‘dbus-1’ found  If you’re facing an error message like:

```
configure: error: Package requirements (dbus-1 >= 1.8) were not met:
No package 'dbus-1' found
```

Consider adjusting the PKG_CONFIG_PATH environment variable if you installed software in a non-standard prefix.

If you’re running a ubuntu system, you may need to run:

```
sudo apt-get install libdbus-glib-1-dev
```

or

```
sudo apt-get install libdbus-1-dev
```

For other linux distros, you may Google the error message and find the solution. This issue could be solved by installing the related header files.

Invalid command ‘bdist_wheel’  If you’re facing an error message like:

```
error: invalid command 'bdist_wheel'
```

You may need to run:

```
python -m pip install -U pip
```

Or

```
python -m pip install wheel
```

Before running the pip commands, please make sure you’re using the IDF python virtual environment.

Basic Concepts

Component-based Unit Tests  Component-based unit tests are our recommended way to test your component. All the test apps should be located under `${IDF_PATH}/components/<COMPONENT_NAME>/test_apps`.

For example:

```
components/
  └── my_component/
      └── include/
      └── test_apps/
          ├── test_app_1
              └── main/
                  └── pytest_my_component_app_1.py
          ├── test_app_2
              └── pytest_my_component_app_2.py
          └── parent_folder
                  └── test_app_3
                      └── pytest_my_component_app_3.py
          └── ...  
      └── my_component.c
          └── CMakeLists.txt
```
Chapter 7. Contributions Guide

Example Tests  Example Tests are tests for examples that are intended to demonstrate parts of the ESP-IDF functionality to our customers.

All the test apps should be located under `${IDF_PATH}/examples`. For more information please refer to the Examples Readme.

For example:

```
examples/
└── parent_folder/
    └── example_1/
        └── main/
            ├── ...
            └── CMakeLists.txt
            └── pytest_example_1.py
```

Custom Tests  Custom Tests are tests that aim to run some arbitrary test internally. They are not intended to demonstrate the ESP-IDF functionality to our customers in any way.

All the test apps should be located under `${IDF_PATH}/tools/test_apps`. For more information please refer to the Custom Test Readme.

Pytest in ESP-IDF

Pytest Execution Process

1. Bootstrapping Phase
   Create session-scoped caches:
   - port-target cache
   - port-app cache

2. Collection Phase
   1. Get all the python files with the prefix `pytest_`
   2. Get all the test functions with the prefix `test_`
   3. Apply the `params`, and duplicate the test functions.
   4. Filter the test cases with CLI options. Introduced detailed usages here

3. Test Running Phase
   1. Construct the fixtures. In ESP-IDF, the common fixtures are initialized in this order:
      1. `pexpect_proc`: pexpect instance
      2. `app`: `IdfApp` instance
         The information of the app, like sdkconfig, flash_files, partition_table, etc., would be parsed at this phase.
      3. `serial`: `IdfSerial` instance
         The port of the host which connected to the target type parsed from the app would be auto-detected.
         The flash files would be auto flashed.
      4. `dut`: `IdfDut` instance
   2. Run the real test function
   3. Deconstruct the fixtures in this order:
      1. `dut`
         1. close the `serial` port
      2. (Only for apps with `unity test framework`) generate junit report of the unity test cases
      3. `serial`
      4. `app`
      4. `pexpect_proc`: Close the file descriptor
   4. (Only for apps with `unity test framework`) Raise `AssertionError` when detected unity test failed if you call `dut.expect_from_unity_output()` in the test function.

4. Reporting Phase
   1. Generate junit report of the test functions
2. Modify the junit report test case name into ESP-IDF test case ID format: `<target>.<config>.<test function name>`.

5. Finalizing Phase (Only for apps with unity test framework)
Combine the junit reports if the junit reports of the unity test cases are generated.

Example Code  This code example is taken from `pytest_console_basic.py`.

```python
@ pytest.mark.esp32
@ pytest.mark.esp32c3
@ pytest.mark.generic
@ pytest.mark.parametrize('config', ['history', 'nohistory'], indirect=True)
def test_console_advanced(config: str, dut: Dut) -> None:
    if config == 'history':
        dut.expect('Command history enabled')
    elif config == 'nohistory':
        dut.expect('Command history disabled')
```

**Note:** Using `expect_exact` is better here. For further reading about the different types of `expect` functions, please refer to the `pytest-embedded Expecting documentation`.

Use Markers to Specify the Supported Targets  You can use markers to specify the supported targets and the test env in CI. You can run `pytest --markers` to get more details about different markers.

```python
@ pytest.mark.esp32  # <-- support esp32
@ pytest.mark.esp32c3  # <-- support esp32c3
@ pytest.mark.generic  # <-- test env 'generic, would assign to runner with tag...
```

Besides, if the test case supports all officially ESP-IDF-supported targets, like esp32, esp32s2, esp32s3, esp32c3 for now (2022.2), you can use a special marker `supported_targets` to apply them all in one line.

This code example is taken from `pytest_gptimer_example.py`.

```python
@ pytest.mark.supported_targets
@ pytest.mark.generic
def test_gptimer_example(dut: Dut) -> None:
    ...
```

Use Params to Specify the sdkconfig Files  You can use `pytest.mark.parametrize` with “config” to apply the same test to different apps with different sdkconfig files. For more information about `sdkconfig.ci.xxx` files, please refer to the Configuration Files section under this readme.

```python
@ pytest.mark.parametrize('config', ['history', 'nohistory'], indirect=True)  # <-- 'indirect=True' is required
```

Overall, this test function would be replicated to 4 test cases:
- esp32.history.test_console_advanced
- esp32.nohistory.test_console_advanced
- esp32c3.history.test_console_advanced
- esp32c3.nohistory.test_console_advanced
**Advanced Examples**

**Multi Dut Tests with the Same App**

```python
@pytest.mark.esp32s2
@ pytest.mark.esp32s3
@ pytest.mark.usb_host
@ pytest.mark.parametrize('count', [2], indirect=True)
def test_usb_host(dut: Tuple[IdfDut, IdfDut]) -> None:
    device = dut[0]  # <-- assume the first dut is the device
    host = dut[1]    # <-- and the second dut is the host
    ...
```

After setting the param `count` to 2, all these fixtures are changed into tuples.

**Multi Dut Tests with Different Apps**  
This code example is taken from `pytest_wifi_getting_started.py`.

```python
@ pytest.mark.esp32
@ pytest.mark.multi_dut_generic
@ pytest.mark.parametrize(
    'count, app_path', [
        (2,  
        f'os.path.join(os.path.dirname(__file__), "softAP") || os.path.join(os.          path.dirname(__file__), "station"))',
    ], indirect=True
)
def test_wifi_getting_started(dut: Tuple[IdfDut, IdfDut]) -> None:
    softap = dut[0]
    station = dut[1]
    ...
```

Here the first dut was flashed with the app `softap`, and the second dut was flashed with the app `station`.

**Multi Dut Tests with Different Apps, and Targets**  
This code example is taken from `pytest_wifi_getting_started.py`. As the comment says, for now it’s not running in the ESP-IDF CI.

```python
@ pytest.mark.parametrize(
    'count, app_path, target', [
        (2,  
        f'os.path.join(os.path.dirname(__file__), "softAP") || os.path.join(os.          path.dirname(__file__), "station"))',
        [esp32|esp32s2],
        (2,  
        f'os.path.join(os.path.dirname(__file__), "softAP") || os.path.join(os.          pathdirname(__file__), "station"))',
        [esp32s2|esp32],
    ], indirect=True
)
def test_wifi_getting_started(dut: Tuple[IdfDut, IdfDut]) -> None:
    softap = dut[0]
    station = dut[1]
    ...
```
Overall, this test function would be replicated to 2 test cases:

- softap with esp32 target, and station with esp32s2 target
- softap with esp32s2 target, and station with esp32 target

Support different targets with different sdkconfig files

This code example is taken from `pytest_panic.py` as an advanced example.

```python
CONFIGS = [
    pytest.param('coredump_flash_bin_crc', marks=[pytest.mark.esp32, pytest.mark.esp32s2]),
    pytest.param('coredump_flash_elf_sha', marks=[pytest.mark.esp32]),  # sha256 only supported on esp32
    pytest.param('coredump_uart_bin_crc', marks=[pytest.mark.esp32, pytest.mark.esp32s2]),
    pytest.param('coredump_uart_elf_crc', marks=[pytest.mark.esp32, pytest.mark.esp32s2]),
    pytest.param('gdbstub', marks=[pytest.mark.esp32, pytest.mark.esp32s2]),
    pytest.param('panic', marks=[pytest.mark.esp32, pytest.mark.esp32s2]),
]
@pytest.mark.parametrize('config', CONFIGS, indirect=True)
...
```

Use Custom Class

Usually, you can write a custom class in these conditions:

1. Add more reusable functions for a certain number of DUTs
2. Add custom setup and teardown functions in different phases described here

This code example is taken from `panic/conftest.py`

```python
class PanicTestDut(IdfDut):
    ...

@ pytest.fixture(scope='module')
def monkeypatch_module(request: FixtureRequest) -> MonkeyPatch:
    mp = MonkeyPatch()
    request.addfinalizer(mp.undo)
    return mp

@ pytest.fixture(scope='module', autouse=True)
def replace_dut_class(monkeypatch_module: MonkeyPatch) -> None:
    monkeypatch_modulesetattr('pytest_embedded_idf.dut.IdfDut', PanicTestDut)
```

monkeypatch_module provide a module-scoped monkeypatch fixture.
replace_dut_class is a module-scoped autouse fixture. This function replaces the IdfDut class with your custom class.

Mark Flaky Tests

Sometimes, our test is based on ethernet or wifi. The network may cause the test flaky. We could mark the single test case within the code repo.

This code example is taken from `pytest_esp_eth.py`

```python
@ pytest.mark.flaky(reruns=3, reruns_delay=5)
def test_esp_eth_ip101(dut: Dut) -> None:
    ...
```

This flaky marker means that if the test function failed, the test case would rerun for a maximum of 3 times with 5 seconds delay.
Mark Known Failure Cases  Sometimes a test couldn’t pass for the following reasons:

- Has a bug
- The success ratio is too low because of environment issue, such as network issue. Retry couldn’t help

Now you may mark this test case with marker xfail with a user-friendly readable reason.

This code example is taken from pytest_panic.py

```python
@ pytest.mark.xfail('config.getvalue("target") == "esp32s2"', reason='raised_→IllegalInstruction instead')
def test_cache_error(dut: PanicTestDut, config: str, test_func_name: str) -> None:

# This marker means that if the test would be a known failure one on esp32s2.
```

Mark Nightly Run Test Cases  Some tests cases are only triggered in nightly run pipelines due to a lack of runners.

```python
@ pytest.mark.nightly_run
```

This marker means that the test case would only be run with env var NIGHTLY_RUN or INCLUDE_NIGHTLY_RUN.

Run the Tests in CI

The workflow in CI is simple, build jobs -> target test jobs.

Build Jobs

Build Job Names

- Component-based Unit Tests: build_pytest_components_<target>
- Example Tests: build_pytest_examples_<target>
- Custom Tests: build_pytest_test_apps_<target>

Build Job Command  The command used by CI to build all the relevant tests is:

```
python $IDF_PATH/tools/ci/ci_build_apps.py <parent_dir> --target <target> -vv --pytest-apps
```

All apps which supported the specified target would be built with all supported sdkconfig files under build_<target>_<config>.

For example, If you run

```
python $IDF_PATH/tools/ci/ci_build_apps.py $IDF_PATH/examples/system/console/basic --target esp32 --pytest-apps
```

the folder structure would be like this:

```
basic
├── build_esp32_history/
│   ├── ...
├── build_esp32_nohistory/
│   └── ...
├── main/
│   └── CMakeLists.txt
├── pytest_console_basic.py
└── ...
```

All the binaries folders would be uploaded as artifacts under the same directories.

Target Test Jobs
Target Test Job Names

- Component-based Unit Tests: component_ut_pytest_<target>_<test_env>
- Example Tests: example_test_pytest_<target>_<test_env>
- Custom Tests: test_app_test_pytest_<target>_<test_env>

Target Test Job Command

The command used by CI to run all the relevant tests is: `pytest <parent_dir> --target <target> -m <test_env_marker>`

All test cases with the specified target marker and the test env marker under the parent folder would be executed. The binaries in the target test jobs are downloaded from build jobs, the artifacts would be placed under the same directories.

Run the Tests Locally

The local executing process is the same as the CI process.

For example, if you want to run all the esp32 tests under the `$IDF_PATH/examples/system/console/basic` folder, you may:

```
$ pip install pytest-embedded-serial-esp pytest-embedded-idf
$ cd $IDF_PATH
$ . ./export.sh
$ cd examples/system/console/basic
$ python $IDF_PATH/tools/ci/ci_build_apps.py . --target esp32 -vv --pytest-apps
$ pytest --target esp32
```

Tips and Tricks

Filter the Test Cases

- filter by target with `pytest --target <target>`
  pytest would run all the test cases that support specified target.
- filter by sdkconfig file with `pytest --sdkconfig <sdkconfig>`
  if `<sdkconfig>` is default, pytest would run all the test cases with the sdkconfig file `sdkconfig`. defaults.
  In other cases, pytest would run all the test cases with `sdkconfig file` `sdkconfig.ci.<sdkconfig>`.

Add New Markers

We’re using two types of custom markers, target markers which indicate that the test cases should support this target, and env markers which indicate that the test case should be assigned to runners with these tags in CI.

You can add new markers by adding one line under the `${IDF_PATH}/pytest.ini markers =` section. The grammar should be: `<marker_name>: <marker_description>`

Generate JUnit Report

You can call pytest with `--junitxml <filepath>` to generate the JUnit report. In ESP-IDF, the test case name would be unified as “<target>.<config>.<function_name>”.

Skip Auto Flash Binary

Skipping auto-flash binary every time would be useful when you’re debugging your test script.

You can call pytest with `--skip-autoflash y` to achieve it.
**Record Statistics**  Sometimes you may need to record some statistics while running the tests, like the performance test statistics.

You can use `record_xml_attribute` fixture in your test script, and the statistics would be recorded as attributes in the JUnit report.

**Logging System**  Sometimes you may need to add some extra logging lines while running the test cases.

You can use `python logging module` to achieve this.

**Known Limitations and Workarounds**

**Avoid Using Thread for Performance Test**  `pytest-embedded` is using some threads internally to help gather all stdout to the pexpect process. Due to the limitation of Global Interpreter Lock, if you’re using threads to do performance tests, these threads would block each other and there would be great performance loss.

**workaround**

Use `Process` instead, the APIs should be almost the same as Thread.

**Further Readings**

- pytest documentation: https://docs.pytest.org/en/latest/contents.html
- pytest-embedded documentation: https://docs.espressif.com/projects/pytest-embedded/en/latest/
Chapter 8

ESP-IDF 版本简介

ESP-IDF 的 GitHub 仓库时常更新，特别是用于开发新特性的 master 分支。如有量产需求，请使用稳定版本。

8.1 发布版本

您可以通过以下链接访问各个版本的配套文档:


ESP-IDF 在 GitHub 平台上的完整发布历史请见 发布说明页面。您可以在该页面查看各个版本的发布说明、配套文档及相应获取方式。

8.2 我该选择哪个版本？

- 如有量产需求，请使用 最新稳定版本。稳定版本已通过人工测试，后续更新仅修复 bug。主要特性不受影响（更多详情，请见版本管理）。请访问 发布说明页面 界面查看每一个稳定发布版本。
- 如需尝试/测试 ESP-IDF 的最新特性，请使用 最新版本 (在 master 分支上)。最新版本包含 ESP-IDF 的所有新特性，已通过自动化测试，但尚未全部完成人工测试 (因此存在一定风险)。
- 如需使用稳定版本中没有的新特性，但同时又不希望受到 master 分支更新的影响，您可以将一个最适合您的稳定版本更新至一个预发布版本 或更新至一个发布分支。
- 如需使用其他基于 ESP-IDF 的项目，请先查看该项目的文档，以确定其兼容的 ESP-IDF 版本。

有关如何更新 ESP-IDF 本地副本的内容，请参考 更新 ESP-IDF 章节。

8.3 版本管理

ESP-IDF 采用了 语义版本管理方法，即您可以从字面含义理解每个版本的差异。其中

- 主要版本 (例 v3.0) 代表有重大更新，包括增加新特性、改变现有特性及移除已弃用的特性。升级至一个新的主要版本 (例 v2.1 升级至 v3.0) 意味着您可能需要更新工程代码，并重新测试工程，具体可参考 发布说明页面 的重大变更 (Breaking Change) 部分。
- 次要版本 (例 v3.1) 代表有新增特性和 bug 修复，但现有特性不受影响，公开 API 的使用也不受影响。
  升级至一个新的次要版本 (例 v3.0 升级至 v3.1) 意味着您可能不需要更新您的工程代码，但需重新测试您的工程，特别是 发布说明页面 中专门提到的部分。
- Bugfix 版本 (例 v3.0.1) 仅修复 bug，并不增加任何新特性。

2517
8.4 支持期限

ESP-IDF 的每个主要版本和次要版本都有相应的支持期限。支持期限满后，版本停止更新维护，将不再提供支持。

支持期限政策 对此有具体描述，并介绍了每个版本的支持期限是如何界定的。

发布说明页面 界面上的每一个发布版本都提供了该版本的支持期限信息。

一般而言：

- 如您刚开始一个新项目，建议使用最新稳定版本。
- 如您有 GitHub 账号，请点击 发布说明页面 界面右上角的“Watch”按键，并选中“Releases only”选项。GitHub 将会在新版本发布的时候通知您。当您所使用的版本有 Bugfix 版本发布时，请做好升级至该 Bugfix 版本的规划。
- 如可能，请定期（如每年一次）将项目的 IDF 版本升级至一个新的主要版本或次要版本。对于次要版本更新，更新过程应该比较简单，但对于主要版本更新，可能需要细致查看发布说明并做对应的更新规划。
- 请确保您所使用的版本停止更新维护前，已做好升级至新版本的规划。

ESP-IDF 的每个主要版本和次要版本（V4.1、V4.2 等）的支持期限为 30 个月，从最初的稳定版发布日算起。

在支持期限内意味着 ESP-IDF 团队将继续在 GitHub 的分支上进行 bug 修复，安全修复等，并根据需要定期发布新的 Bugfix 版本。

支持期限分为“服务期”和“维护期”：

<table>
<thead>
<tr>
<th>周期</th>
<th>时长</th>
<th>是否推荐新工程使用</th>
</tr>
</thead>
<tbody>
<tr>
<td>服务期</td>
<td>12 个月</td>
<td>是</td>
</tr>
<tr>
<td>维护期</td>
<td>18 个月</td>
<td>否</td>
</tr>
</tbody>
</table>

在服务期内，Bugfix 版本的发布更为频繁。某些情况下，在服务期内会增加新特性，这些特性主要是为了满足新产品特定监管要求或标准，并且回归风险非常低。

在维护期内，该版本仍受支持，但只会对严重性较高的问题或安全问题进行 bug 修复。

当开始一个新项目时，建议使用在服务期内的版本。

鼓励用户在您所用的版本支持期限结束之前，将所有的工程升级到最新的 ESP-IDF 版本。在版本支持期限满后，我们将不再继续进行 bug 修复。

支持期限包括预发布版本（betas，预览版，-rc 和 -dev 版等），有时会将某个特性在发布版中标记为“预览版”，这意味着该特性不在支持期限内。

关于 不同版本的 ESP-IDF （主要版本、次要版本、Bugfix 版本等）的更多信息，请参考 ESP-IDF 编程指南。
8.5 查看当前版本

查看 ESP-IDF 本地副本的版本，请使用 idf.py 命令:

```
idf.py --version
```

此外，由于 ESP-IDF 的版本也已编译至固件中，因此您也可以使用宏 IDF_VER 查看 ESP-IDF 的版本 (以字符串的格式)。ESP-IDF 默认引导程序会在设备启动时打印 ESP-IDF 的版本，请注意，在 GitHub 仓库中的代码更新时，代码中的版本信息仅会在源代码重新编译或在清除编译时才会更新，因此打印出来的版本可能并不是最新的。

如果编写的代码需要支持多个 ESP-IDF 版本，可以在编译时使用 compile-time macros 检查版本。

几个 ESP-IDF 版本的例子:

<table>
<thead>
<tr>
<th>版本字符串</th>
<th>含义</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3.2-dev-306-gbeb3611ca</td>
<td>master 分支上的预发布版本。</td>
</tr>
<tr>
<td></td>
<td>- v3.2-dev: 为 v3.2 进行的开发。</td>
</tr>
<tr>
<td></td>
<td>- 306: v3.2 开发启动后的 commit 数量。</td>
</tr>
<tr>
<td></td>
<td>- geb3611ca: commit 标识符。</td>
</tr>
<tr>
<td>v3.0.2</td>
<td>稳定版本，标签为 v3.0.2。</td>
</tr>
<tr>
<td>v3.1-beta1-75-g346d6b0ea</td>
<td>v3.1 的 beta 测试版本 (可参考更新至一个发布分支)。</td>
</tr>
<tr>
<td></td>
<td>- v3.1-beta1: 预发布标签。</td>
</tr>
<tr>
<td></td>
<td>- 75: 添加预发布 beta 标签后的 commit 数量。</td>
</tr>
<tr>
<td></td>
<td>- g346d6b0ea: commit 标识符。</td>
</tr>
<tr>
<td>v3.0.1-dirty</td>
<td>稳定版本，标签为 v3.0.1。</td>
</tr>
<tr>
<td></td>
<td>- dirty 代表 ESP-IDF 的本地副本有修改。</td>
</tr>
</tbody>
</table>
8.6 Git 工作流

乐鑫 ESP-IDF 团队的 (Git) 开发工作流程如下:

- 新的改动总是在 master 分支（最新版本）进行。master 分支上的 ESP-IDF 版本总带有 -dev 标签，表示“正在开发中”。例如 v3.1-dev。
- 这些改动将首先在乐鑫的内部 Git 仓库进行代码审阅与测试，而后在自动化测试完成后推至 GitHub。
- 新版本一旦完成特性开发（在 master 分支上进行）并达到进入 Beta 测试的标准，则将该版本切换至一个新分支（例如 release/v3.1）。此外，该分支还打上预发布标签（例如 v3.1-beta1）。您可以在 GitHub 平台上查看 ESP-IDF 的完整分支列表和标签列表。Beta 预发布版本可能仍存在大量“已知问题”（Known Issue）。
- 随着对 Beta 版本的不断测试，bug 修复将同时增加至该发布分支和 master 分支。而且，master 分支可能也已经开始为下个版本开发新特性了。
- 当测试结束时，该发布分支上将增加一个 `rc` 标签，代表候选发布（Release Candidate），如 `v3.1-rc1`。此时，该分支仍属于预发布版本。
- 如果一直未发现或报告重大 bug，则该预发布版本将最终增加“主要版本”（例 v4.0）或“次要版本”标记（例 v3.1），成为正式发布版本，并体现在发布说明页面。
- 后续，发布版本中发现的 bug 都将在该发布分支上进行修复。
- 发布分支上会定期进行 bug 修复，人工测试完成后，该分支将增加一个 Bugfix 版本标签（例如 v3.1.1），并体现在发布说明页面。

8.7 更新 ESP-IDF

请根据您的实际情况，对 ESP-IDF 进行更新。

- 如有量产用途，建议参考更新至一个稳定发布版本。
- 如需测试/研发/尝试最新特性，建议参考更新至 master 分支。
- 两者折衷建议参考更新至一个发布分支。

备注：在参考本指南时，请首先获得 ESP-IDF 的本地副本，具体步骤请参考入门指南中的介绍。

8.7.1 更新至一个稳定发布版本

对于量产用户，推荐更新至一个新的 ESP-IDF 发布版本，请参考以下步骤：

- 请定期查看 发布说明页面，了解最新发布情况。
- 如有新发布的 Bugfix 版本（如 v3.0.1 或 v3.0.2）时，请将新的 Bugfix 版本更新至您的 ESP-IDF 目录。
- 在 Linux 或 macOS 系统中，请运行如下命令将分支更新至 vX.Y.Z:

```
cd $IDF_PATH
git fetch

git checkout vX.Y.Z

git submodule update --init --recursive
```

- 在 Windows 系统中，需要将 cd $IDF_PATH 替换为 cd %IDF_PATH%。
- 在主要版本或次要版本新发布时，请查看发布说明中的具体描述，并决定是否升级您的版本。具体命令与上述描述一致。

备注：如果您之前在安装 ESP-IDF 时使用了 zip 文件包，而非通过 Git 命令，则您将无法使用 Git 命令进行版本升级，此属正常情况。这种情况下，请重新下载最新 zip 文件包，并替换掉之前 IDF_PATH 下的全部内容。
8.7.2 更新至一个预发布版本

您可以将您的本地副本切换（命令git checkout）至一个预发布版本或rc版本，具体方法请参考更新至一个稳定发布版本中的描述。

预发布版本通常不体现在发布说明页面。更多详情，请查看完整标签列表。使用预发布版本的注意事项，请参考更新至一个发布分支中的描述。

8.7.3 更新至master分支

备注：ESP-IDF中master分支上的代码会时时更新，因此使用master分支相当在“流血的边缘试探”，存在一定风险。

如需使用ESP-IDF的master分支，请参考以下步骤：

- 在Linux或macOS系统中，使用如下命令在本地切换至master分支：

```bash
cd $IDF_PATH
git checkout master
git pull
git submodule update --init --recursive
```

- 在Windows系统中，需要将cd $IDF_PATH替换为cd %IDF_PATH%。

此外，您还需要在后续工作中不时使用git pull命令，将远端master上的更新同步到本地。注意，在更新master分支后，您可能需要更改工程代码，也可能遇到新的bug。
- 如需从master分支切换至一个发布分支或稳定版本，请使用git checkout命令。

重要：强烈建议您定期使用git pull和git submodule update --init --recursive命令，确保本地副本得到及时更新。旧的master分支相当于一个“快照”，可能存在未记录的问题，且无法获得支持。对于半稳定版本，请参考更新至一个发布分支。

8.7.4 更新至一个发布分支

从稳定性来说，使用“发布分支”相当于在使用master分支和稳定版本之间进行折衷，包含一些master分支上的新特性，但也同时保证可通过beta测试且基本完成了bug修复。

更多详情，请前往GitHub查看完整标签列表。

例如，在Linux或macOS系统中，您可以运行以下命令更新至ESP-IDF v3.1，随时关注该分支上的Bugfix版本发布（如v3.1.1等）：

```bash
cd $IDF_PATH
git fetch
git checkout release/v3.1
git pull
git submodule update --init --recursive
```

在Windows系统中，需要将cd $IDF_PATH替换为cd %IDF_PATH%。

您每次在该分支上使用git pull时都相当于把最新的Bugfix版本发布更新至您的本地副本中。

备注：发布分支并不会有专门的配套文档，建议您使用与本分支最接近版本的文档。
Chapter 9

资源

9.1 PlatformIO

• What is PlatformIO?
• Installation
• Configuration
• Tutorials
• Project Examples
• Next Steps

9.1.1 What is PlatformIO?

PlatformIO is a cross-platform embedded development environment with out-of-the-box support for ESP-IDF.

Since ESP-IDF support within PlatformIO is not maintained by the Espressif team, please report any issues with PlatformIO directly to its developers in the official PlatformIO repositories.

A detailed overview of the PlatformIO ecosystem and its philosophy can be found in the official PlatformIO documentation.

9.1.2 Installation

• PlatformIO IDE is a toolset for embedded C/C++ development available on Windows, macOS and Linux platforms
• PlatformIO Core (CLI) is a command-line tool that consists of multi-platform build system, platform and library managers and other integration components. It can be used with a variety of code development environments and allows integration with cloud platforms and web services
Chapter 9. 资源

9.1.3 Configuration

Please go through the official PlatformIO configuration guide for ESP-IDF.

9.1.4 Tutorials

- ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis

9.1.5 Project Examples

Please check ESP-IDF page in the official PlatformIO documentation

9.1.6 Next Steps

Here are some useful links for exploring the PlatformIO ecosystem:

- Learn more about integrations with other IDEs/Text Editors
- Get help from PlatformIO community

9.2 有用的链接

- 您可以在 ESP32 论坛 中提出您的问题，访问社区资源。
- 您可以通过 GitHub 的 Issues 版块提交 bug 或功能请求。在提交新 Issue 之前，请先查看现有的 Issues。
- 您可以在 ESP IoT Solution 库中找到基于 ESP-IDF 的解决方案、应用实例、组件和驱动等内容。多数文档均提供中英文版本。
- 通过 Arduino 平台开发应用，请参考 ESP32、ESP32-S2 和 ESP32-C3 芯片的 Arduino 内核。
- 关于 ESP32 的书籍列表，请查看 乐鑫 网站。
- 如果您有兴趣参与到 ESP-IDF 的开发，请查阅 Contributions Guide。
- 关于 ESP32 的其它信息，请查看官网 文档 版块。
- 关于本文档的 PDF 和 HTML 格式下载（最新版本和早期版本），请点击 下载。
Chapter 10

Copyrights and Licenses

10.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2022 Espressif Systems. This source code is licensed under the Apache License 2.0 as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses.

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.

Some examples use external components which are not Apache licensed, please check the copyright description in each example source code.

10.1.1 Firmware Components

These third party libraries can be included into the application (firmware) produced by ESP-IDF.

- **Newlib** is licensed under the BSD License and is Copyright of various parties, as described in COPYING.NEWLIB.
- **Xtensa header files** are Copyright (C) 2013 Tensilica Inc and are licensed under the MIT License as reproduced in the individual header files.
- Original parts of **FreeRTOS** (components/freertos) are Copyright (C) 2017 Amazon.com, Inc. or its affiliates are licensed under the MIT License, as described in license.txt.
- Original parts of **LWIP** (components/lwip) are Copyright (C) 2001, 2002 Swedish Institute of Computer Science and are licensed under the BSD License as described in COPYING file.
- **wpa_supplicant** Copyright (c) 2003-2005 Jouni Malinen and licensed under the BSD license.
- **FreeBSD net80211** Copyright (c) 2004-2008 Sam Leffler, Erno Consulting and licensed under the BSD license.
- **argtable3** argument parsing library Copyright (C) 1998-2001,2003-2011,2013 Stewart Heitmann and licensed under 3-clause BSD license. argtable3 also includes the following software components. For details, please see argtable3 LICENSE file.
  - C Hash Table library, Copyright (c) 2002, Christopher Clark and licensed under 3-clause BSD license.
  - The Better String library, Copyright (c) 2014, Paul Hsieh and licensed under 3-clause BSD license.
  - TCL library, Copyright the Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, ActiveState Corporation and other parties, and licensed under TCL/TK License.
- **linenoise** line editing library Copyright (c) 2010-2014 Salvatore Sanfilippo, Copyright (c) 2010-2013 Pieter Noordhuis, licensed under 2-clause BSD license.
- **FatFS** library, Copyright (C) 2017 ChaN, is licensed under a BSD-style license.
- ** cJSON** library, Copyright (c) 2009-2017 Dave Gamble and cJSON contributors, is licensed under MIT license as described in LICENSE file.
- **micro-ecc** library, Copyright (c) 2014 Kenneth MacKay, is licensed under 2-clause BSD license.
• **Mbed TLS library**, Copyright (C) 2006-2018 ARM Limited, is licensed under Apache License 2.0 as described in LICENSE file.
• **SPIFFS library**, Copyright (c) 2013-2017 Peter Andersson, is licensed under MIT license as described in LICENSE file.
• **SD/MMC driver** is derived from OpenBSD SD/MMC driver, Copyright (c) 2006 Uwe Stuehler, and is licensed under BSD license.
• **ESP-MQTT** MQTT Package (contiki-mqtt) - Copyright (c) 2014, Stephen Robinson, MQTT-ESP - Tuan PM (<tuanpm at live dot com>) is licensed under Apache License 2.0 as described in LICENSE file.
• **BLE Mesh** is adapted from Zephyr Project, Copyright (c) 2017-2018 Intel Corporation and licensed under Apache License 2.0
• **mynewt-nimble** Apache Mynewt NimBLE, Copyright 2015-2018, The Apache Software Foundation, is licensed under Apache License 2.0 as described in LICENSE file.
• **TLSF allocator** <https://github.com/espressif/tlsf> Two Level Segregated Fit memory allocator, Copyright (c) 2006-2016, Matthew Conte, and licensed under the BSD 3-clause license.
• **openthread** Copyright (c) The OpenThread Authors, is licensed under BSD License as described in LICENSE file.
• **UBSAN runtime** — Copyright (c) 2016, Linaro Limited and Jiří Zárevůcký, licensed under the BSD 2-clause license.
• **HTTP Parser** Based on src/http/ngx_http_parse.c from NGINX copyright Igor Sysoev. Additional changes are licensed under the same terms as NGINX and Joyent, Inc. and other Node contributors. For details please check LICENSE file.
• **SEGGER SystemView** target-side library, Copyright (c) 2015-2017 SEGGER Microcontroller GmbH & Co. KG, is licensed under BSD 3-clause license.

### 10.1.2 Documentation

• HTML version of the ESP-IDF Programming Guide uses the Sphinx theme sphinx_idf_theme, which is Copyright (c) 2013-2020 Dave Snider, Read the Docs, Inc. & contributors, and Espressif Systems (Shanghai) CO., LTD. It is based on sphinx_rtd_theme. Both are licensed under MIT license.

### 10.2 ROM Source Code Copyrights

ESP32, ESP32-S and ESP32-C Series SoCs mask ROM hardware includes binaries compiled from portions of the following third party software:

• **Newlib** , licensed under the BSD License and is Copyright of various parties, as described in COPYING.NEWLIB.
• **Xtensa libhal**, Copyright (c) Tensilica Inc and licensed under the MIT license (see below).
• **TinyBasic** Plus, Copyright Mike Field & Scott Lawrence and licensed under the MIT license (see below).
• **miniz**, by Rich Geldreich - placed into the public domain.
• **wpa_supplicant** Copyright (c) 2003-2005 Jouni Malinen and licensed under the BSD license.
• **TJpgDec** Copyright (C) 2011, ChaN, all right reserved. See below for license.

### 10.3 Xtensa libhal MIT License

Copyright (c) 2003, 2006, 2010 Tensilica Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
Chapter 10. Copyrights and Licenses

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

10.4 TinyBasic Plus MIT License

Copyright (c) 2012-2013

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

10.5 TJpgDec License

TJpgDec - Tiny JPEG Decompressor R0.01 (C)ChaN, 2011 The TJpgDec is a generic JPEG decompressor module for tiny embedded systems. This is a free software that opened for education, research and commercial developments under license policy of following terms.

Copyright (C) 2011, ChaN, all right reserved.

- The TJpgDec module is a free software and there is NO WARRANTY.
- No restriction on use. You can use, modify and redistribute it for personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
- Redistributions of source code must retain the above copyright notice.
Chapter 11

关于本指南

本指南为乐鑫公司 ESP32 官方应用开发框架 ESP-IDF 的配套文档。

ESP32 芯片是一款 2.4 GHz Wi-Fi 和蓝牙双模芯片，内置 1 或 2 个 Xtensa® 32 位 LX6 处理器，运算能力最高可达 600 DMIPS。

ESP-IDF 即乐鑫物联网开发框架，可为在 Windows、Linux 和 macOS 系统平台上开发 ESP32 应用程序工具链、API、组件和工作流程的支持。

图 1: 乐鑫物联网综合开发框架
Chapter 11. 关于本指南
Chapter 12

切换语言

《ESP-IDF 编程指南》部分文档现在有两种语言的版本。如有出入请以英文版本为准。

- 英文
- 中文

如下图所示，如果该文档两种语言版本均具备，您可以通过点击文档上方的语言链接轻松进行语言切换。

Fatal Errors

[中文]

Overview

In certain situations, execution of the program can not be continued in a well defined way. In ESP-IDF, these situations include:
索引

符号

- _ESP_LOG_EARLY_ENABLED (C macro), 1995
- _ip_addr (C++ struct), 952
- _ip_addr::ip4 (C++ member), 952
- _ip_addr::ip6 (C++ member), 952
- _ip_addr::type (C++ member), 952
- _ip_addr::u_addr (C++ member), 952
- [anonymous] (C++ enum), 469, 491, 569, 1715, 2049
- [anonymous]::ESP_BLE_MESH_SERVER_FLAG_MAX (C++ enumerator), 569
- [anonymous]::ESP_BLE_MESH_SERVER_TRANS_TIMER_START (C++ enumerator), 569
- [anonymous]::ESP_BLE_SCA_100PPM (C++ enumerator), 491
- [anonymous]::ESP_BLE_SCA_150PPM (C++ enumerator), 491
- [anonymous]::ESP_BLE_SCA_200PPM (C++ enumerator), 491
- [anonymous]::ESP_BLE_SCA_250PPM (C++ enumerator), 491
- [anonymous]::ESP_BLE_SCA_300PPM (C++ enumerator), 492
- [anonymous]::ESP_BLE_SCA_500PPM (C++ enumerator), 491
- [anonymous]::ESP_BLE_SCA_750PPM (C++ enumerator), 492
- [anonymous]::ESP_ERR_FLASH_NO_RESPONSE (C++ enumerator), 1715
- [anonymous]::ESP_ERR_FLASH_SIZE_NOT_MATCH (C++ enumerator), 1715
- [anonymous]::ESP_ERR_SLEEP_REJECT (C++ enumerator), 2049
- [anonymous]::ESP_ERR_SLEEP TOO_SHORT_SLEEP_DURATION (C++ enumerator), 2049
- [anonymous]::ESP_HIDD_BOOT_REPORT_SIZE_KEYBOARD (C++ enumerator), 469
- [anonymous]::ESP_HIDD_BOOT_REPORT_SIZE_MOUSE (C++ enumerator), 470

A

adc_atten_t (C++ enum), 964
adc_atten_t::ADC_ATTEN_DB_0 (C++ enumerator), 964
adc_atten_t::ADC_ATTEN_DB_11 (C++ enumerator), 965
adc_atten_t::ADC_ATTEN_DB_2_5 (C++ enumerator), 965
adc_atten_t::ADC_ATTEN_DB_6 (C++ enumerator), 965
adc_bitwidth_t (C++ enum), 965
adc_bitwidth_t::ADC_BITWIDTH_10 (C++ enumerator), 965
adc_bitwidth_t::ADC_BITWIDTH_11 (C++ enumerator), 965
adc_bitwidth_t::ADC_BITWIDTH_12 (C++ enumerator), 965
adc_bitwidth_t::ADC_BITWIDTH_13 (C++ enumerator), 965
adc_bitwidth_t::ADC_BITWIDTH_DEFAULT (C++ enumerator), 965
adc_calib_check_scheme (C++ function), 978
adc_calib_handle_t (C++ type), 979
adc_calib_raw_to_voltage (C++ function), 978
adc_calib_scheme_ver_t (C++ enum), 979
adc_calib_scheme_ver_t::ADC_CALI_SCHEME_VER_CURVE_FITTING (C++ enumerator), 979
adc_calib_scheme_ver_t::ADC_CALI_SCHEME_VER_LINE_FITTING (C++ enumerator), 979
adc_channel_t (C++ enum), 964
adc_channel_t::ADC_CHANNEL_0 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_1 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_2 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_3 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_4 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_5 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_6 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_7 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_8 (C++ enumerator), 964
adc_channel_t::ADC_CHANNEL_9 (C++ enumerator), 964
adc_continuous_config_t::conv_mode (C++ member), 975
dc_continuous_config_t::format (C++ member), 975
dc_continuous_config_t::pattern_num (C++ member), 975
dc_continuous_config_t::sample_freq_hz (C++ member), 975
dc_continuous_deinit (C++ function), 974
dc_continuous_evt_cbs_t (C++ struct), 975
dc_continuous_evt_cbs_t::on_conv_done (C++ member), 976
dc_continuous_evt_cbs_t::on_pool_ovf (C++ member), 976
dc_continuous_evt_data_t (C++ struct), 975
dc_continuous_evt_data_t::conv_frame (C++ member), 975
dc_continuous_evt_data_t::size (C++ member), 975
dc_continuous_handle_cfg_t (C++ struct), 974
dc_continuous_handle_cfg_t::conv_frame (C++ member), 975
dc_continuous_handle_cfg_t::max_store (C++ member), 974
dc_continuous_handle_t (C++ type), 976
dc_continuous_io_to_channel (C++ function), 974
dc_continuous_new_handle (C++ function), 972
dc_continuous_read (C++ function), 973
dc_continuous_register_event_callback (C++ function), 973
dc_continuous_start (C++ function), 973
dc_continuous_stop (C++ function), 974
dc_digi_convert_mode_t (C++ enum), 965
dc_digi_convert_mode_t::ADC_CONV_ALTER_UNIT (C++ enumerator), 966
dc_digi_convert_mode_t::ADC_CONV_BOTH_UNITS (C++ enumerator), 966
dc_digi_convert_mode_t::ADC_CONV_DISABLE (C++ enumerator), 966
dc_digi_convert_mode_t::ADC_CONV_DISABLE (C++ struct), 963
dc_digi_convert_mode_t::ADC_CONV_DISABLE (C++ member), 963
adc_digi_output_data_t::type2 (C++ member), 963
adc_digi_output_data_t::unit (C++ member), 963
adc_digi_output_data_t::val (C++ member), 963
adc_digi_output_format_t::ADC_DIGI_OUTPUT_FORMAT_TYPE2 (C++ struct), 962
adc_digi_output_format_t::ADC_DIGI_OUTPUT_FORMAT_TYPE1 (C++ struct), 962
adc_digi_pattern_config_t (C++ struct), 962
adc_digi_pattern_config_t::atten (C++ member), 963
adc_digi_pattern_config_t::bit_width (C++ member), 963
adc_digi_pattern_config_t::channel (C++ member), 963
adc_digi_pattern_config_t::unit (C++ member), 963
ADC_MAX_DELAY (C macro), 976
adc_oneshot_channel_cfg_t (C++ struct), 968
adc_oneshot_new_unit (C++ function), 966
adc_oneshot_new_unit (C++ struct), 966
adc_oneshot_register_event_callback (C++ function), 966
adc_oneshot_stop (C++ function), 974
adc_ulp_mode_t (C++ enum), 965
adc_ulp_mode_t::ADC_ULP_MODE_DISABLE (C++ dictator), 965
adc_ulp_mode_t::ADC_ULP_MODE_DISABLE (C++ struct), 964
adc_unit_t (C++ member), 963
ADD_DEV_FLUSHABLE_DEV_FLAG (C macro), 541
ADD_DEV_RM_AFTER_PROV_FLAG (C macro), 541
ADD_DEV_START_PROV_NOW_FLAG (C macro), 541
E

EFD_SUPPORT_ISR (C macro), 1745
efuse_hal_chip_revision (C++ function), 1764
efuse_hal_get_mac (C++ function), 1764
efuse_hal_get_major_chip_version (C++ function), 1764
efuse_hal_get_minor_chip_version (C++ function), 1764
emac_rmii_clock_gpio_t (C++ enum), 911
emac_rmii_clock_gpio_t:EMAC_CLK_OUT_GPIO (C++ enumerator), 911
emac_rmii_clock_gpio_t:EMAC_CLK_IN_GPIO (C++ enumerator), 911
emac_rmii_clock_gpio_t:EMAC_CLK_OUT_180_GPIO (C++ enumerator), 911
emac_rmii_clock_gpio_t:EMAC_CLK_OUT_GPIO (C++ enumerator), 911
emac_rmii_clock_mode_t (C++ enum), 911
emac_rmii_clock_mode_t:EMAC_CLK_DEFAULT (C++ enumerator), 911
emac_rmii_clock_mode_t:EMAC_CLK_EXT_IN (C++ enumerator), 911
emac_rmii_clock_mode_t:EMAC_CLK_OUT (C++ enumerator), 911
eNotifyAction (C++ enum), 1850
eNotifyAction::eIncrement (C++ enumerator), 1850
eNotifyAction::eSetValueWithoutOverwrite (C++ enumerator), 1850
eNotifyAction::eSetValueWithOverwrite (C++ enumerator), 1850
eSleepModeStatus (C++ enum), 1850
eSleepModeStatus::eAbortSleep (C++ enumerator), 1850
eSleepModeStatus::eNoTasksWaitingTimeout (C++ enumerator), 1850
eSleepModeStatus::eStandardSleep (C++ enumerator), 1850
esp_a2d_audio_state_t (C++ enum), 391
esp_a2d_audio_state_t:ESP_A2D_AUDIO_STATE_REMOTE_SUSPEND (C++ enumerator), 391
esp_a2d_audio_state_t:ESP_A2D_AUDIO_STATE_REMOTE_STOPPED (C++ enumerator), 391
esp_a2d_event_t (C++ enum), 393
esp_a2d_menu_param_t::a2d_audio_cfg_param::mcc (C++ member), 387
esp_a2d_menu_param_t::a2d_audio_cfg_param::remote_bda (C++ member), 387
esp_a2d_menu_param_t::a2d_audio_stat_param (C++ struct), 387
esp_a2d_menu_param_t::a2d_audio_stat_param::remote_bda (C++ member), 388
esp_a2d_menu_param_t::a2d_conn_stat_param (C++ struct), 388
esp_a2d_menu_param_t::a2d_conn_stat_param::remote_bda (C++ member), 388
esp_a2d_menu_param_t::a2d_menu_param_t (C++ class), 387

Espressif Systems
Submit Document Feedback
esp_a2d_sink_set_delay_value (C++ function), 385
esp_a2d_source_connect (C++ function), 386
esp_a2d_source_data_cb_t (C++ type), 391
esp_a2d_source_deinit (C++ function), 386
esp_a2d_source_disconnect (C++ function), 386
esp_a2d_source_init (C++ function), 386
esp_a2d_source_register_data_callback (C++ function), 386
esp_alloc_failed_hook_t (C++ type), 1955
ESP_APP_DESC_MAGIC_WORD (C macro), 2012
esp_app_desc_t (C++ struct), 2011
esp_app_desc_t::app_elf_sha256 (C++ member), 2012
esp_app_desc_t::date (C++ member), 2012
esp_app_desc_t::idf_ver (C++ member), 2012
esp_app_desc_t::magic_word (C++ member), 2012
esp_app_desc_t::project_name (C++ member), 2012
esp_app_desc_t::reserv1 (C++ member), 2012
esp_app_desc_t::reserv2 (C++ member), 2012
esp_app_desc_t::secure_version (C++ member), 2012
esp_app_desc_t::time (C++ member), 2012
esp_app_desc_t::version (C++ member), 2012
esp_app_get_description (C++ function), 2011
esp_app_get_elf_sha256 (C++ function), 2011
ESP_APP_ID_MAX (C macro), 238
ESP_APP_ID_MIN (C macro), 238
esp_apptrace_buffer_get (C++ function), 1756
esp_apptrace_buffer_put (C++ function), 1756
esp_apptrace_dest_t (C++ enum), 1759
esp_apptrace_dest_t::ESP_APPTRACE_DEST_ESP32_A2D_CPU (C++ enum), 1759
esp_apptrace_dest_t::ESP_APPTRACE_DEST_ESP32_A2D_HOST (C++ enum), 1759
esp_apptrace_dest_t::ESP_APPTRACE_DEST_UART (C++ enum), 1759
esp_apptrace_dest_t::ESP_APPTRACE_DEST_TRAX (C++ enum), 1759
esp_apptrace_dest_t::ESP_APPTRACE_DEST_JTAG (C++ enum), 1759
esp_apptrace_dest_t::ESP_APPTRACE_DEST_NUM (C++ enum), 1759
esp_apptrace_flush (C++ function), 1757
esp_apptrace_flush_nolock (C++ function), 1757
esp_apptrace_fopen (C++ function), 1758
esp_apptrace_fread (C++ function), 1758
esp_apptrace_fseek (C++ function), 1758
esp_apptrace_fstop (C++ function), 1759
esp_apptrace_ftell (C++ function), 1758
esp_apptrace_fwrite (C++ function), 1758
esp_apptrace_host_is_connected (C++ function), 1758
esp_apptrace_init (C++ function), 1755
esp_apptrace_read (C++ function), 1757
esp_apptrace_vprintf (C++ function), 1756
esp_apptrace_vprintf_to (C++ function), 1756
esp_apptrace_write (C++ function), 1756
esp_attr_control_t (C++ struct), 305
esp_attr_control_t::auto_rsp (C++ member), 306
esp_attr_desc_t (C++ struct), 305
esp_attr_desc_t::length (C++ member), 305
esp_attr_desc_t::max_length (C++ member), 305
esp_attr_desc_t::perm (C++ member), 305
esp_attr_desc_t::uuid_length (C++ member), 305
esp_attr_desc_t::uuid_p (C++ member), 305
esp_attr_desc_t::value (C++ member), 305
esp_attr_value_t (C++ struct), 306
esp_attr_value_t::attr_len (C++ member), 306
esp_attr_value_t::attr_max_len (C++ member), 306
esp_attr_value_t::attr_value (C++ member), 306
esp_avrc_batt_stat_t (C++ enum), 414
esp_avrc_batt_stat_t::ESP_AVRC_BATT_CRITICAL (C++ enumerator), 414
esp_avrc_batt_stat_t::ESP_AVRC_BATT_EXTERNAL (C++ enumerator), 414
esp_avrc_batt_stat_t::ESP_AVRC_BATT_FULL_CHARGE (C++ enumerator), 414
esp_avrc_batt_stat_t::ESP_AVRC_BATT_NORMAL (C++ enumerator), 414
esp_avrc_batt_stat_t::ESP_AVRC_BATT_WARNING (C++ enumerator), 414
dx_avrc_bit_mask_op_t (C++ enum), 408
dx_avrc_bit_mask_op_t::ESP_AVRC_BIT_MASK_OP_CLEAR (C++ enumerator), 409
dx_avrc_bit_mask_op_t::ESP_AVRC_BIT_MASK_OP_SET (C++ enumerator), 409
dx_avrc_bit_mask_op_t::ESP_AVRC_BIT_MASK_OP_TEST (C++ enumerator), 408
dx_avrc_ct_cb_event_t (C++ enum), 409
dx_avrc_ct_cb_event_t::ESP_AVRC_CT_CONNECTION_STATE_EVT (C++ enumerator), 409
dx_avrc_ct_cb_event_t::ESP_AVRC_CT_CHANGE_NOTIFY (C++ enumerator), 409
dx_avrc_ct_cb_event_t::ESP_AVRC_CT_CONNECTION_START (C++ enumerator), 409

Espresif Systems 2539 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
Esperrif Systems

Esperrif Systems

esp_avrc_tg_deinit (C++ function), 396
esp_avrc_tg_get_psth_cmd_filter (C++ function), 396
esp_avrc_tg_get_rn_evt_cap (C++ function), 397
esp_avrc_tg_init (C++ function), 396
esp_avrc_tg_register_callback (C++ function), 396
esp_avrc_tg_send_rn_rsp (C++ function), 398
esp_avrc_tg_set_psth_cmd_filter (C++ function), 396
esp_avrc_tg_set_rn_evt_cap (C++ function), 397
ESP_AVRC_TRANS_LABEL_MAX (C macro), 403
esp_base_mac_addr_get (C++ function), 2003
esp_base_mac_addr_set (C++ function), 2003
ESP_BD_ADDR_HEX (C macro), 239
ESP_BD_ADDR_LEN (C macro), 238
ESP_BD_ADDR_STR (C macro), 239
esp_bd_addr_t (C++ type), 239
esp_ble_addr_type_t (C++ enum), 240
esp_ble_addr_type_t::BLE_ADDR_TYPE_PUBLIC (C++ enum), 240
esp_ble_addr_type_t::BLE_ADDR_TYPE_RANDOM (C++ enum), 240
esp_ble_addr_type_t::BLE_ADDR_TYPE_RPA (C++ enum), 240
esp_ble_addr_type_t::BLE_ADDR_TYPE_RPA_CMP (C++ enum), 240
esp_ble_adv_channel_t (C++ enum), 299
esp_ble_adv_channel_t::ADV_CHNL_37 (C++ enum), 299
esp_ble_adv_channel_t::ADV_CHNL_38 (C++ enum), 299
esp_ble_adv_channel_t::ADV_CHNL_39 (C++ enum), 299
esp_ble_adv_channel_t::ADV_CHNL_ALL (C++ enum), 299
ESP_BLE_ADV_DATA_LEN_MAX (C macro), 289
esp_ble_adv_data_t (C++ struct), 271
esp_ble_adv_data_t::appearance (C++ member), 271
esp_ble_adv_data_t::flag (C++ member), 272
esp_ble_adv_data_t::include_name (C++ member), 271
esp_ble_adv_data_t::include_txpower (C++ member), 271
esp_ble_adv_data_t::manufacturer_len (C++ member), 271
esp_ble_adv_data_t::max_interval (C++ member), 271
esp_ble_adv_data_t::min_interval (C++ member), 271
esp_ble_adv_data_t::p_manufacturer_data (C++ member), 271
esp_ble_adv_data_t::p_service_data (C++ member), 271
esp_ble_adv_data_t::p_service_uuid (C++ member), 272
esp_ble_adv_data_t::service_data_len (C++ member), 271
esp_ble_adv_data_t::service_uuid_len (C++ member), 272
esp_ble_adv_data_t::set_scan_rsp (C++ member), 271
esp_ble_adv_data_type (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_MANUFACTURER_SPECIFIC (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_128SERVICE_DATA (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_128SRV_CMP (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_128SRV_PART (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_16SRV_CMPL (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_16SRV_PART (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_32SRV_CMPL (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_32SRV_PART (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_32SERVICE_DATA (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_32SERVICE_DATA (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_32SRV_CMP (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_32SRV_PART (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_ADV_INT (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_ADV_INT (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_APPEARANCE (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_CHAN_MAP_UPDATE (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_DEV_CLASS (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_DEV_CLASS (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_FLAG (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_INDOOR_POS (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_INT_RANGE (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_LE_DEV_ADD (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_LE_ROLE (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_LE_SECURE (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_LE_SECURE (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_LE_SUPPORT (C++ enum), 298
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_NAME_CMPL (C++ enum), 297
esp_ble_adv_data_type::ESP_BLE_AD_TYPE_NAME_SHORT (C++ enum), 297

Submit Document Feedback
esp_ble_adv_params_t::peer_addr

esp_ble_adv_params_t::own_addr_type

esp_ble_adv_params_t::channel_map

esp_ble_adv_params_t::adv_type

esp_ble_adv_params_t::adv_int_min

esp_ble_adv_params_t::adv_filter_policy

esp_ble_adv_params_t::adv_type

esp_ble_adv_params_t::channel_map

esp_ble_adv_params_t::own_addr_type

esp_ble_adv_params_t::peer_addr

ESP_BLE_ADV_FLAG_NON_LIMIT_DISC

ESP_BLE_ADV_FLAG_LIMIT_DISC

ESP_BLE_ADV_FLAG_GEN_DISC

ESP_BLE_ADV_FLAG_DMT_HOST_SPT

ESP_BLE_ADV_FLAG_BREDR_NOT_SPT

esp_ble_adv_filter_t::ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST

esp_ble_adv_filter_t::ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY

esp_ble_adv_filter_t::ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_URI

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_TX_PWR

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_TRANS_DISC_DATA

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_SPAIR_R256

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_SPAIR_C256

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_SOL_SRV_UUID

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_SM_TK

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_SM_OOB

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_SM_OOB

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_PUBLIC_TARGET

esp_ble_adv_data_type::ESP_BLE_AD_TYPE_RANDOM_TARGET

ESP_BLE_APPEARANCE_GENERIC_INSULIN_PUMP

ESP_BLE_APPEARANCE_GENERIC_HEART_RATE

ESP_BLE_APPEARANCE_GENERIC_GLUCOSE

ESP_BLE_APPEARANCE_GENERIC_EYEGLASSES

ESP_BLE_APPEARANCE_GENERIC_DISPLAY

ESP_BLE_APPEARANCE_GENERIC_CYCLING

ESP_BLE_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MONITOR

ESP_BLE_APPEARANCE_GENERIC_CLOCK

ESP_BLE_APPEARANCE_GENERIC_BLOOD_PRESSURE

ESP_BLE_APPEARANCE_GENERIC_BARCODE_SCANNER

ESP_BLE_ADV_REPORT_EXT_SCAN_RSP

ESP_BLE_ADV_REPORT_EXT_DIRECT_ADV

ESP_BLE_ADV_REPORT_EXT_ADV_IND

ESP_BLE_ADV_REPORT_EXT_ADV_IND

ESP_BLE_ADV_REPORT_EXT_DIRECT_IND_HIGH

ESP_BLE_ADV_REPORT_EXT_DIRECT_IND_LOW

ESP_BLE_ADV_REPORT_EXT_DIRECT_IND

ESP_BLE_ADV_TYPE_SCAN_IND

ESP_BLE_ADV_TYPE_NONCONN_IND

ESP_BLE_ADV_TYPE_IND

ESP_BLE_ADV_TYPE_DIRECT_IND

ESP_BLE_ADV_TYPE_DIRECT_IND

ESP_BLE_ADV_TYPE_SCAN_IND

ESP_BLE_ADV_TYPE_NONCONN_IND

ESP_BLE_ADV_TYPE_IND

ESP_BLE_ADV_TYPE_DIRECT_IND

ESP_BLE_ADV_TYPE_DIRECT_IND

ESP_BLE_ADV_TYPE_DIRECT_IND

ESP_BLE_ADV_TYPE_DIRECT_IND

ESP_BLE_ADV_TYPE_DIRECT_IND

ESP_BLE_ADV_APPEARANCE_BLOOD_PRESSURE_ARM

ESP_BLE_ADV_APPEARANCE_BLOOD_PRESSURE_WRIST

ESP_BLE_ADV_APPEARANCE_CYCLING_CADENCE

ESP_BLE_ADV_APPEARANCE_CYCLING_COMPUTER

ESP_BLE_ADV_APPEARANCE_CYCLING_POWER

ESP_BLE_ADV_APPEARANCE_CYCLING_SPEED

ESP_BLE_ADV_APPEARANCE_CYCLING_SPEED_CADENCE

ESP_BLE_ADV_APPEARANCE_GENERIC_BARCODE_SCANNER

ESP_BLE_ADV_APPEARANCE_GENERIC_BLOOD_PRESSURE

ESP_BLE_ADV_APPEARANCE_GENERIC_CLOCK

ESP_BLE_ADV_APPEARANCE_GENERIC_COMPUTER

ESP_BLE_ADV_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MON

ESP_BLE_ADV_APPEARANCE_GENERIC_CYCLING

ESP_BLE_ADV_APPEARANCE_GENERIC_DISPLAY

ESP_BLE_ADV_APPEARANCE_GENERIC_EYEGLASSES

ESP_BLE_ADV_APPEARANCE_GENERIC_GLUCOSE

ESP_BLE_ADV_APPEARANCE_GENERIC_HEART_RATE

ESP_BLE_ADV_APPEARANCE_GENERIC_HID

ESP_BLE_ADV_APPEARANCE_GENERIC_INSULIN_PUMP
<table>
<thead>
<tr>
<th>C++ struct</th>
<th>C++ member</th>
</tr>
</thead>
<tbody>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_data_set_cmpl_param::status</td>
<td>260</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_data_set_cmpl_param::adv_phy</td>
<td>260</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_data_set_cmpl_param::adv_clk_accuracy</td>
<td>260</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_data_set_cmpl_param::adv_addr_type</td>
<td>260</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_data_set_cmpl_param::adv_addr</td>
<td>259</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_start_cmpl_param::status</td>
<td>260</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_sync_estab_param::sid</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_sync_estab_param::period_adv_interval</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_sync_estab_param::adv_phy</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_sync_estab_param::adv_clk_accuracy</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_sync_estab_param::adv_addr_type</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_sync_estab_param::adv_addr</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_stop_cmpl_param::status</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_start_cmpl_param::status</td>
<td>261</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_sync_terminate_cmpl_param::status</td>
<td>262</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_cancel_cmpl_param::status</td>
<td>262</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_create_sync_cmpl_param::status</td>
<td>262</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_clear_dev_cmpl_param::status</td>
<td>262</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_add_dev_cmpl_param::status</td>
<td>262</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_data_set_cmpl_param::status</td>
<td>263</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_report_param::params</td>
<td>263</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_data_set_cmpl_param::adv_phy</td>
<td>263</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_data_set_cmpl_param::adv_clk_accuracy</td>
<td>263</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_data_set_cmpl_param::adv_addr_type</td>
<td>263</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_periodic_adv_data_set_cmpl_param::adv_addr</td>
<td>263</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_set_db_adv_cfg_event_param::ble_period_adv_createsync_event::status</td>
<td>264</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_set_db_adv_cfg_event_param::ble_period_adv_endevent::status</td>
<td>264</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_set_db_adv_cfg_event_param::ble_period_adv_startevent::status</td>
<td>264</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_set_db_adv_cfg_event_param::ble_period_adv_stopevent::status</td>
<td>264</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_set_db_adv_cfg_event_param::ble_period_adv_cleardevevent::status</td>
<td>264</td>
</tr>
<tr>
<td>esp_ble_gap_cb_param_t::ble_ext_adv_set_db_adv_cfg_event_param::ble_period_adv_adddevevent::status</td>
<td>264</td>
</tr>
</tbody>
</table>

Submit Document Feedback
esp_ble_gap_conn_params_t::scan_interval
esp_ble_gap_conn_params_t::min_ce_len
esp_ble_gap_conn_params_t::max_ce_len
esp_ble_gap_conn_params_t::latency
esp_ble_gap_conn_params_t::interval_min

esp_ble_gap_ext_adv_params_t::type
esp_ble_gap_ext_adv_params_t::tx_power
esp_ble_gap_ext_adv_params_t::sid
esp_ble_gap_ext_adv_params_t::secondary_phy
esp_ble_gap_ext_adv_params_t::scan_req_notif
esp_ble_gap_ext_adv_data_status_t
ESP_BLE_GAP_EXT_ADV_DATA_COMPLETE
ESP_BLE_GAP_EXT_ADV_DATA_INCOMPLETE
ESP_BLE_GAP_EXT_ADV_DATA_TRUNCATED

esp_ble_gap_clear_whitelist
esp_ble_gap_clean_duplicate_scan_exceptional_list
esp_ble_gap_ext_adv_reprot_t::dir_addr
esp_ble_gap_ext_adv_reprot_t::data_status
esp_ble_gap_ext_adv_reprot_t::adv_data_len
esp_ble_gap_ext_adv_reprot_t::adv_data
esp_ble_gap_ext_adv_reprot_t::addr_type
esp_ble_gap_ext_adv_reprot_t::addr

esp_ble_gap_conn_params_t::supervision_timeout
esp_ble_gap_conn_params_t::scan_window
esp_ble_gap_conn_params_t::interval_max
esp_ble_gap_conn_params_t::min_ce_len
esp_ble_gap_conn_params_t::max_ce_len
esp_ble_gap_conn_params_t::scan_interval
esp_ble_gap_conn_params_t::interval_min
None
esp_ble_local_id_keys_t::irk (C++ member), 277
ESP_BLE_MESH_ACTUATOR_BLOCKED_ERROR (C macro), 630
ESP_BLE_MESH_ACTUATOR_BLOCKED_WARNING (C macro), 630
ESP_BLE_MESH_ADDR_ALL_NODES (C macro), 537
ESP_BLE_MESH_ADDR_FRIENDS (C macro), 537
ESP_BLE_MESH_ADDR_IS_GROUP (C macro), 539
ESP_BLE_MESH_ADDR_IS_RFU (C macro), 539
ESP_BLE_MESH_ADDR_IS_UNICAST (C macro), 539
ESP_BLE_MESH_ADDR_IS_VIRTUAL (C macro), 539
ESP_BLE_MESH_ADDR_PROXIES (C macro), 537
ESP_BLE_MESH_ADDR_RELAYS (C macro), 538
ESP_BLE_MESH_ADDR_TYPE_PUBLIC (C macro), 541
ESP_BLE_MESH_ADDR_TYPE_RANDOM (C macro), 541
ESP_BLE_MESH_ADDR_TYPE_RPA_PUBLIC (C macro), 541
ESP_BLE_MESH_ADDR_TYPE_RPA_RANDOM (C macro), 541
esp_ble_mesh_addr_type_t (C++ type), 560
ESP_BLE_MESH_ADDR_UNASSIGNED (C macro), 537
ESP_BLE_MESH_BATTERY_LOW_ERROR (C macro), 629
ESP_BLE_MESH_BATTERY_LOW_WARNING (C macro), 629
esp_ble_mesh_bd_addr_t (C++ type), 560
ESP_BLE_MESH_BEACON_DISABLED (C macro), 538
ESP_BLE_MESH_BEACON_ENABLED (C macro), 538
esp_ble_mesh_cb_t (C++ mem-ber), 277
esp_ble_mesh_cfg_app_key_add_t::app_idx (C++ member), 601
esp_ble_mesh_cfg_app_key_add_t::app_key (C++ member), 601
esp_ble_mesh_cfg_app_key_add_t::net_idx (C++ member), 601
esp_ble_mesh_cfg_app_key_delete_t (C++ struct), 606
esp_ble_mesh_cfg_app_key_delete_t::app_idx (C++ member), 606
esp_ble_mesh_cfg_app_key_delete_t::net_idx (C++ member), 606
esp_ble_mesh_cfg_app_key_get_t (C++ struct), 599
esp_ble_mesh_cfg_app_key_get_t::net_idx (C++ member), 599
esp_ble_mesh_cfg_app_key_list_cb_t (C++ struct), 614
esp_ble_mesh_cfg_app_key_list_cb_t::app_idx (C++ member), 614
esp_ble_mesh_cfg_app_key_list_cb_t::net_idx (C++ member), 614
esp_ble_mesh_cfg_app_key_list_cb_t::status (C++ member), 614
esp_ble_mesh_cfg_app_key_status_cb_t (C++ struct), 611
esp_ble_mesh_cfg_app_key_status_cb_t::app_idx (C++ member), 611
esp_ble_mesh_cfg_app_key_status_cb_t::net_idx (C++ member), 611
esp_ble_mesh_cfg_app_key_status_cb_t::status (C++ member), 611
esp_ble_mesh_cfg_app_key_update_t (C++ struct), 606
esp_ble_mesh_cfg_app_key_update_t::app_idx (C++ member), 606
esp_ble_mesh_cfg_app_key_update_t::app_key (C++ member), 606
esp_ble_mesh_cfg_app_key_update_t::net_idx (C++ member), 606
app_cfg_ble_mesh_cfg_beacon_set.t (C++ struct), 600
app_cfg_ble_mesh_cfg_beacon_set.t::beacon (C++ member), 600
app_cfg_ble_mesh_cfg_beacon_set.t::beacon_status (C++ struct), 608
app_cfg_ble_mesh_cfg_beacon_set.t::beacon_status (C++ member), 609
app_cfg_ble_mesh_cfg_beacon_set.t::client_cb_event (C++ enum), 621
app_cfg_ble_mesh_cfg_beacon_set.t::client_cb_event (C++ member), 621
app_cfg_ble_mesh_cfg_beacon_set.t::client_cb_event (C++ enumerator), 621
app_cfg_ble_mesh_cfg_client_cb_event (C++ struct), 601
app_cfg_ble_mesh_cfg_client_cb_event (C++ member), 621
esp_ble_mesh_cfg_client_cb_t::esp_ble_mesh_cfg_client_get_state_t::app_key_get
(C++ member), 591
esp_ble_mesh_cfg_client_cb_param_t::esp_ble_mesh_cfg_client_get_state_t::comp_data_get
(C++ member), 591
esp_ble_mesh_cfg_client_cb_param_t::esp_ble_mesh_cfg_client_get_state_t::kr_phase_get
(C++ member), 592
esp_ble_mesh_cfg_client_cb_param_t::esp_ble_mesh_cfg_client_get_state_t::ltn_pollto_get
(C++ member), 592
esp_ble_mesh_cfg_client_cb_param_t::esp_ble_mesh_cfg_client_get_state_t::model_pub_get
(C++ member), 591
esp_ble_mesh_cfg_client_common_cb_param_t::node_identity_status
esp_ble_mesh_cfg_client_common_cb_param_t::netkey_status
esp_ble_mesh_cfg_client_common_cb_param_t::netkey_list
esp_ble_mesh_cfg_client_common_cb_param_t::net_transmit_status
esp_ble_mesh_cfg_client_common_cb_param_t::model_sub_status
esp_ble_mesh_cfg_client_common_cb_param_t::model_sub_list
esp_ble_mesh_cfg_client_common_cb_param_t::model_pub_status
esp_ble_mesh_cfg_client_common_cb_param_t::model_app_status
esp_ble_mesh_cfg_client_common_cb_param_t::model_app_list
esp_ble_mesh_cfg_client_common_cb_param_t::lpn_timeout_status
esp_ble_mesh_cfg_client_common_cb_param_t::kr_phase_status
esp_ble_mesh_cfg_client_common_cb_param_t::heartbeat_sub_status
esp_ble_mesh_cfg_client_common_cb_param_t::gatt_proxy_status
esp_ble_mesh_cfg_client_common_cb_param_t::friend_status
esp_ble_mesh_cfg_client_common_cb_param_t::default_ttl_status
esp_ble_mesh_cfg_client_common_cb_param_t::appkey_status
esp_ble_mesh_cfg_client_common_cb_param_t::appkey_list
esp_ble_mesh_cfg_client_common_cb_param_t::app_key_update
esp_ble_mesh_cfg_client_common_cb_param_t::app_key_delete
esp_ble_mesh_cfg_client_common_cb_param_t::app_key_add
esp_ble_mesh_cfg_client_set_state_t::model_sub_overwrite
esp_ble_mesh_cfg_client_set_state_t::model_sub_delete_all
esp_ble_mesh_cfg_client_set_state_t::model_sub_delete
esp_ble_mesh_cfg_client_set_state_t::model_sub_add
esp_ble_mesh_cfg_client_set_state_t::model_pub_va_set
esp_ble_mesh_cfg_client_set_state_t::model_pub_set
esp_ble_mesh_cfg_client_set_state_t::model_app_unbind
esp_ble_mesh_cfg_client_set_state_t::model_app_bind
esp_ble_mesh_cfg_client_set_state_t::kr_phase_set
esp_ble_mesh_cfg_client_set_state_t::heartbeat_sub_set
esp_ble_mesh_cfg_client_set_state_t::heartbeat_pub_set
esp_ble_mesh_cfg_client_set_state_t::gatt_proxy_set
esp_ble_mesh_cfg_client_set_state_t::friend_set
esp_ble_mesh_cfg_client_set_state_t::default_ttl_set
esp_ble_mesh_cfg_client_set_state_t::app_key_update
esp_ble_mesh_cfg_client_set_state_t::app_key_delete
esp_ble_mesh_cfg_client_get_state_t::vnd_model_sub_get
esp_ble_mesh_cfg_client_get_state_t::vnd_model_app_get
esp_ble_mesh_cfg_client_get_state_t::sig_model_sub_get
esp_ble_mesh_cfg_client_get_state_t::sig_model_app_get
esp_ble_mesh_cfg_client_get_state_t::node_identity_get
esp_ble_mesh_cfg_client_get_state_t::model_pub_get
esp_ble_mesh_cfg_client_get_state_t::lpn_pollto_get
esp_ble_mesh_cfg_client_get_state_t::kr_phase_get
esp_ble_mesh_cfg_client_get_state_t::app_key_get
esp_ble_mesh_cfg_client_common_cb_param_t::model_pub_status
esp_ble_mesh_cfg_client_common_cb_param_t::model_app_status
esp_ble_mesh_cfg_client_common_cb_param_t::model_app_list
esp_ble_mesh_cfg_client_common_cb_param_t::lpn_timeout_status
esp_ble_mesh_cfg_client_common_cb_param_t::kr_phase_status
esp_ble_mesh_cfg_client_common_cb_param_t::heartbeat_sub_status
esp_ble_mesh_cfg_client_common_cb_param_t::gatt_proxy_status
esp_ble_mesh_cfg_client_common_cb_param_t::friend_status
esp_ble_mesh_cfg_client_common_cb_param_t::default_ttl_status
esp_ble_mesh_cfg_client_common_cb_param_t::appkey_status
esp_ble_mesh_cfg_client_common_cb_param_t::appkey_list
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_pub_status_cb_t::features
(C++ member), 593
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_pub_status_cb_t::net_idx
(C++ member), 593
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_pub_status_cb_t::period
(C++ member), 593
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_pub_status_cb_t::status
(C++ member), 592
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_sub_status_cb_t::status
(C++ member), 593
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_sub_status_cb_t::src
(C++ member), 593
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_sub_status_cb_t::period
(C++ member), 593
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_sub_status_cb_t::min_hops
(C++ member), 593
esp_ble_mesh_cfg_client_set_state_t::model
esp_ble_mesh_cfg_hb_sub_status_cb_t::max_hops
(C++ member), 593
esp_ble_mesh_cfg_comp_data_status_cb_t esp_ble_mesh_cfg_hb_sub_status_cb_t::max_hops
(C++ member), 613
esp_ble_mesh_cfg_comp_data_status_cb_t esp_ble_mesh_cfg_hb_sub_status_cb_t::min_hops
(C++ member), 613
esp_ble_mesh_cfg_comp_data_status_cb_t esp_ble_mesh_cfg_hb_sub_status_cb_t::period
(C++ member), 609
esp_ble_mesh_cfg_comp_data_status_cb_t esp_ble_mesh_cfg_hb_sub_status_cb_t::src
(C++ member), 609
esp_ble_mesh_cfg_composition_data_get_testesp_ble_mesh_cfg_hb_sub_status_cb_t::status
(C++ member), 598
esp_ble_mesh_cfg_composition_data_get_testesp_ble_mesh_cfg_heartbeat_pub_set_t
(C++ struct), 608
esp_ble_mesh_cfg_default_ttl_set_t esp_ble_mesh_cfg_heartbeat_pub_set_t::count
(C++ member), 600
esp_ble_mesh_cfg_default_ttl_set_t esp_ble_mesh_cfg_heartbeat_sub_set_t::tls
(C++ member), 600
esp_ble_mesh_cfg_default_ttl_set_t esp_ble_mesh_cfg_heartbeat_sub_set_t::dst
(C++ member), 600
esp_ble_mesh_cfg_default_ttl_set_t esp_ble_mesh_cfg_heartbeat_sub_set_t::feature
(C++ member), 600
esp_ble_mesh_cfg_default_ttl_set_t esp_ble_mesh_cfg_heartbeat_sub_set_t::net_idx
(C++ member), 600
esp_ble_mesh_cfg_friend_set_t esp_ble_mesh_cfg_heartbeat_pub_set_t::period
(C++ member), 600
esp_ble_mesh_cfg_friend_set_t esp_ble_mesh_cfg_heartbeat_pub_set_t::period
(C++ member), 600
esp_ble_mesh_cfg_friend_set_t esp_ble_mesh_cfg_heartbeat_pub_set_t::period
(C++ member), 600
esp_ble_mesh_cfg_friend_status_cb_t esp_ble_mesh_cfg_heartbeat_sub_set_t
(C++ struct), 612
esp_ble_mesh_cfg_friend_status_cb_t esp_ble_mesh_cfg_heartbeat_sub_set_t::dst
(C++ member), 612
esp_ble_mesh_cfg_gatt_proxy_set_t esp_ble_mesh_cfg_heartbeat_sub_set_t::period
(C++ member), 601
esp_ble_mesh_cfg_gatt_proxy_set_t esp_ble_mesh_cfg_heartbeat_sub_set_t::src
(C++ member), 609
esp_ble_mesh_cfg_gatt_proxy_status_cb_t esp_ble_mesh_cfg_hr phase_get_t
(C++ struct), 600
esp_ble_mesh_cfg_gatt_proxy_status_cb_t esp_ble_mesh_cfg_hr phase_get_t::net_idx
(C++ member), 600
esp_ble_mesh_cfg_hr pub status_cb_t esp_ble_mesh_cfg_hr phase_set_t
(C++ struct), 607
esp_ble_mesh_cfg_hr pub status_cb_t esp_ble_mesh_cfg_hr phase_set_t::net_idx
(C++ member), 607
esp_ble_mesh_cfg_hr pub status_cb_t esp_ble_mesh_cfg_hr phase_set_t::transition
(C++ member), 612

Espressif Systems 2561 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
esp_ble_mesh_cfg_net_key_add_t::net_key (C++ member), 605
esp_ble_mesh_cfg_net_key_add_t::net_idx (C++ member), 603
esp_ble_mesh_cfg_net_key_add_t::company_id (C++ member), 603
esp_ble_mesh_cfg_net_key_add_t::model_id (C++ member), 603
esp_ble_mesh_cfg_net_key_add_t::label_uuid (C++ member), 603
esp_ble_mesh_cfg_net_key_add_t::element_addr (C++ member), 603
esp_ble_mesh_cfg_net_key_add_t::cred_flag (C++ member), 603
esp_ble_mesh_cfg_net_key_add_t (C++ struct), 601

esp_ble_mesh_cfg_net_key_delete_t (C++ struct), 606

esp_ble_mesh_cfg_model_sub_overwrite_t::company_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t::model_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t::element_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t::label_uuid (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_delete_t::model_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t::label_uuid (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t::element_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t::company_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t::status (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_add_t (C++ struct), 602

esp_ble_mesh_cfg_model_sub_add_t::company_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_add_t::model_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_add_t::label_uuid (C++ member), 605
esp_ble_mesh_cfg_model_sub_add_t::element_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_add_t::cred_flag (C++ member), 605
esp_ble_mesh_cfg_model_sub_add_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_status_cb_t::sub_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_status_cb_t::status (C++ member), 605
esp_ble_mesh_cfg_model_sub_status_cb_t::model_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_status_cb_t::element_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_status_cb_t::company_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_status_cb_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_overwrite_t::sub_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t::model_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t::element_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t::company_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_overwrite_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_delete_t::sub_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t::model_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t::element_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t::company_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_delete_all_t::model_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_all_t::element_addr (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_all_t::company_id (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_all_t::status (C++ member), 605
esp_ble_mesh_cfg_model_sub_delete_all_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_add_t::company_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::model_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::label_uuid (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::element_addr (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::cred_flag (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_add_t::company_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::model_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::label_uuid (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::element_addr (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::cred_flag (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_add_t::company_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::model_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::label_uuid (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::element_addr (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t::cred_flag (C++ member), 603
esp_ble_mesh_cfg_model_sub_add_t (C++ struct), 601

esp_ble_mesh_cfg_net_key_delete_t (C++ struct), 606

esp_ble_mesh_cfg_net_key_delete_t::net_key (C++ member), 605
esp_ble_mesh_cfg_net_key_delete_t::net_idx (C++ member), 605
esp_ble_mesh_cfg_net_key_delete_t::model_id (C++ member), 605
esp_ble_mesh_cfg_net_key_delete_t::label_uuid (C++ member), 605
esp_ble_mesh_cfg_net_key_delete_t (C++ struct), 601

esp_ble_mesh_cfg_model_sub_overwrite_t::model_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_overwrite_t::element_addr (C++ member), 603
esp_ble_mesh_cfg_model_sub_overwrite_t::company_id (C++ member), 603
esp_ble_mesh_cfg_model_sub_overwrite_t::status (C++ member), 603
esp_ble_mesh_cfg_model_sub_overwrite_t (C++ struct), 601
esp_ble_mesh_cfg_relay_status_cb_t::retransmit

esp_ble_mesh_cfg_relay_status_cb_t::relay

esp_ble_mesh_cfg_relay_set_t::relay

esp_ble_mesh_cfg_relay_status_cb_t::relay

esp_ble_mesh_cfg_node_identity_set_t::net_idx

esp_ble_mesh_cfg_node_identity_set_t::identity

esp_ble_mesh_cfg_node_identity_get_t::net_idx

esp_ble_mesh_cfg_node_id_status_cb_t::net_idx

esp_ble_mesh_cfg_node_id_status_cb_t::identity

esp_ble_mesh_cfg_net_trans_set_t::net_transmit

esp_ble_mesh_cfg_net_trans_status_cb_t::net_trans_step

esp_ble_mesh_cfg_net_trans_status_cb_t::net_trans_count

esp_ble_mesh_cfg_net_key_update_t::net_key

esp_ble_mesh_cfg_net_key_update_t::net_idx

esp_ble_mesh_cfg_net_key_status_cb_t::status

esp_ble_mesh_cfg_net_key_status_cb_t::net_idx

esp_ble_mesh_cfg_net_key_list_cb_t::net_idx

esp_ble_mesh_cfg_net_key_delete_t::net_idx

esp_ble_mesh_cfg_node_id_status_cb_t::net_idx

esp_ble_mesh_cfg_sig_model_sub_get_t::model_id

esp_ble_mesh_cfg_sig_model_sub_get_t::element_addr

esp_ble_mesh_cfg_sig_model_app_get_t::model_id

esp_ble_mesh_cfg_server_state_change_t::netkey_delete

esp_ble_mesh_cfg_server_state_change_t::netkey_add

esp_ble_mesh_cfg_server_state_change_t::mod_sub_delete

esp_ble_mesh_cfg_server_state_change_t::mod_sub_add

esp_ble_mesh_cfg_server_state_change_t::mod_pub_set

esp_ble_mesh_cfg_server_state_change_t::mod_app_unbind

esp_ble_mesh_cfg_server_state_change_t::mod_app_bind

esp_ble_mesh_cfg_server_state_change_t::kr_phase_set

esp_ble_mesh_cfg_server_state_change_t::appkey_update

esp_ble_mesh_cfg_server_state_change_t::appkey_delete

esp_ble_mesh_cfg_server_state_change_t::appkey_add
esp_ble_mesh_cfg_srv (C++ struct), 596
esp_ble_mesh_cfg_srv::beacon (C++ member), 597
esp_ble_mesh_cfg_srv::count (C++ member), 597
esp_ble_mesh_cfg_srv::default_ttl (C++ member), 597
esp_ble_mesh_cfg_srv::dst (C++ member), 597
esp_ble_mesh_cfg_srv::expiry (C++ member), 597
esp_ble_mesh_cfg_srv::feature (C++ member), 597
esp_ble_mesh_cfg_srv::friend_state (C++ member), 597
esp_ble_mesh_cfg_srv::gatt_proxy (C++ member), 597
esp_ble_mesh_cfg_srv::heartbeat_pub (C++ member), 597
esp_ble_mesh_cfg_srv::heartbeat_recv_cb (C++ member), 598
esp_ble_mesh_cfg_srv::heartbeat_sub (C++ member), 598
esp_ble_mesh_cfg_srv::max_hops (C++ member), 597
esp_ble_mesh_cfg_srv::min_hops (C++ member), 597
esp_ble_mesh_cfg_srv::model (C++ member), 596
esp_ble_mesh_cfg_srv::net_idx (C++ member), 597
esp_ble_mesh_cfg_srv::net_transmit (C++ member), 596
esp_ble_mesh_cfg_srv::period (C++ member), 597
esp_ble_mesh_cfg_srv::relay (C++ member), 596
esp_ble_mesh_cfg_srv::relay_retransmit (C++ member), 596
esp_ble_mesh_cfg_srv::src (C++ member), 597
esp_ble_mesh_cfg_srv::timer (C++ member), 597
esp_ble_mesh_cfg_srv::ttl (C++ member), 597
esp_ble_mesh_cfg_srv_t (C++ type), 620
ESP_BLE_MESH_CFG_STATUS_INVALID_ADDRESS (C macro), 548
ESP_BLE_MESH_CFG_STATUS_INVALID_APPKEY (C macro), 548
ESP_BLE_MESH_CFG_STATUS_INVALID_BINDING (C macro), 549
ESP_BLE_MESH_CFG_STATUS_INVALID_MODEL (C macro), 548
ESP_BLE_MESH_CFG_STATUS_INVALID_NETKEY (C macro), 548
ESP_BLE_MESH_CFG_STATUS_INVALID_PUBLISH_PARAMETERS (C macro), 548
ESP_BLE_MESH_CFG_STATUS_KEY_INDEX_ALREADY_STORED (C macro), 548
ESP_BLE_MESH_CFG_STATUS_NOT_A_SUBSCRIBE_MODEL (C macro), 548
ESP_BLE_MESH_CFG_STATUS_STORAGE_FAILURE (C macro), 549
ESP_BLE_MESH_CFG_STATUS_SUCCESS (C macro), 548
esp_ble_mesh_cfg_status_t (C++ type), 560
ESP_BLE_MESH_CFG_STATUS_TEMP_UNABLE_TO_CHANGE_STATE (C macro), 549
ESP_BLE_MESH_CFG_STATUS_UNSPECIFIED_ERROR (C macro), 549
esp_ble_mesh_cfg_vnd_model_app_get_t (C++ struct), 599
esp_ble_mesh_cfg_vnd_model_app_get_t::company_id (C++ member), 600
esp_ble_mesh_cfg_vnd_model_app_get_t::element_addr (C++ member), 599
esp_ble_mesh_cfg_vnd_model_app_get_t::model_id (C++ member), 599
esp_ble_mesh_cfg_vnd_model_sub_get_t (C++ struct), 598
esp_ble_mesh_cfg_vnd_model_sub_get_t::company_id (C++ member), 599
esp_ble_mesh_cfg_vnd_model_sub_get_t::element_addr (C++ member), 599
esp_ble_mesh_cfg_vnd_model_sub_get_t::model_id (C++ member), 599
esp_ble_mesh_client_common_param_t (C++ struct), 534
esp_ble_mesh_client_common_param_t::ctx (C++ member), 535
esp_ble_mesh_client_common_param_t::model (C++ member), 535
esp_ble_mesh_client_common_param_t::msg_role (C++ member), 535
esp_ble_mesh_client_common_param_t::msg_timeout (C++ member), 535
esp_ble_mesh_client_common_param_t::opcode (C++ member), 535
esp_ble_mesh_client_common_param_t::op (C++ member), 535
ESP_BLE_MESH_CLIENT_MODEL_DEINIT (C++ function), 575
ESP_BLE_MESH_CLIENT_MODEL_INIT (C++ function), 575

esp_ble_mesh_client_model_send_msg

esp_ble_mesh_client_op_pair_t

esp_ble_mesh_client_op_pair_t::cli_op

esp_ble_mesh_client_op_pair_t::status

esp_ble_mesh_client_t

esp_ble_mesh_client_t::model

esp_ble_mesh_client_t::msg_role

esp_ble_mesh_client_t::op_pair

esp_ble_mesh_client_t::op_pair_size

esp_ble_mesh_client_t::publish_status

esp_ble_mesh_comp_t

esp_ble_mesh_comp_t::cid

esp_ble_mesh_comp_t::element_count

esp_ble_mesh_comp_t::elements

esp_ble_mesh_comp_t::vid

ESP_BLE_MESH_CONDITIONAL_ERROR

ESP_BLE_MESH_CONDITIONAL_WARNING

esp_ble_mesh_config_client_get_state

esp_ble_mesh_config_client_set_state

ESP_BLE_MESH_CONFIGURATION_ERROR

ESP_BLE_MESH_CONFIGURATION_WARNING

esp_ble_mesh_deinit

esp_ble_mesh_deinit_param_t::erase_flash

esp_ble_mesh_deinit_param_t::flags

esp_ble_mesh_deinit_param_t::uuid

esp_ble_mesh_device_delete_t

esp_ble_mesh_device_delete_t::addr

esp_ble_mesh_device_delete_t::addr_type

esp_ble_mesh_device_delete_t::flag

esp_ble_mesh_device_delete_t::uid

ESP_BLE_MESH_DEVICE_DROPPED_ERROR

ESP_BLE_MESH_DEVICE_DROPPED_WARNING

ESP_BLE_MESH_DEVICE_MOVED_ERROR

ESP_BLE_MESH_DEVICE_MOVED_WARNING

ESP_BLE_MESH_DEVICE_NAME_MAX_LEN

esp_ble_mesh_elem_t

esp_ble_mesh_elem_t::location

esp_ble_mesh_elem_t::model

esp_ble_mesh_elem_t::sig_model_count

esp_ble_mesh_elem_t::sig_models

esp_ble_mesh_elem_t::vnd_model_count

esp_ble_mesh_elem_t::vnd_models

ESP_BLE_MESH_ELEMENT

ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_ERROR

ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_WARNING

ESP_BLE_MESH_EMPTY_ERROR

ESP_BLE_MESH_EMPTY_WARNING

esp_ble_mesh_fast_prov_action_t

esp_ble_mesh_fast_prov_action_t::FAST_PROV_ACT_NONE

esp_ble_mesh_fast_prov_action_t::FAST_PROV_ACT_ENTER

esp_ble_mesh_fast_prov_action_t::FAST_PROV_ACT_EXIT

esp_ble_mesh_fast_prov_action_t::FAST_PROV_ACT_MAX

esp_ble_mesh_fast_prov_action_t::FAST_PROV_ACT_SUSPEND

esp_ble_mesh_fast_prov_info_t

esp_ble_mesh_fast_prov_info_t::iv_index

esp_ble_mesh_fast_prov_info_t::flags

esp_ble_mesh_fast_prov_info_t::iv_index

esp_ble_mesh_fast_prov_info_t::iv_index

esp_ble_mesh_fast_prov_info_t::iv_index

ESP_BLE_MESH_GENERIC_ERROR

ESP_BLE_MESH_GENERIC_WARNING

ESP_BLE_MESH_INVALID_ARGS_ERROR
Espresif Systems

Submit Document Feedback
esp_ble_mesh_gen_client_prop_srv_t::model
(C++ member), 657
esp_ble_mesh_gen_def_trans_time_srv_t::model
(C++ member), 651
esp_ble_mesh_gen_client_prop_srv_t::rsp_ctrl
(C++ member), 657
esp_ble_mesh_gen_def_trans_time_srv_t::state
(C++ member), 657
esp_ble_mesh_gen_client_properties_get_rsp_ble_mesh_gen_def_trans_time_state_t
(C++ struct), 643
esp_ble_mesh_gen_client_properties_get_rsp_ble_mesh_gen_def_trans_time_state_t::trans_time
(C++ member), 651
esp_ble_mesh_gen_client_properties_get_rsp_ble_mesh_gen_def_trans_time_state_t::trans_time
(C++ member), 651
esp_ble_mesh_gen_client_properties_status_cb_t::user_property_status
(C++ struct), 644
esp_ble_mesh_gen_client_properties_status_cb_t::power_range_status
(C++ struct), 648
esp_ble_mesh_gen_client_properties_status_cb_t::power_last_status
(C++ struct), 649
esp_ble_mesh_gen_client_properties_status_cb_t::power_default_status
(C++ struct), 644
esp_ble_mesh_gen_client_properties_status_cb_t::onpowerup_status
(C++ struct), 644
esp_ble_mesh_gen_client_properties_status_cb_t::onoff_status
(C++ struct), 644
esp_ble_mesh_gen_client_properties_status_cb_t::manufacturer_property_status
(C++ struct), 644
esp_ble_mesh_gen_client_properties_status_cb_t::manufacturer_properties_status
(C++ struct), 644
esp_ble_mesh_gen_client_properties_status_cb_t::location_local_status
(C++ struct), 644
esp_ble_mesh_gen_def_trans_time_status_cb_t::trans_time
(C++ member), 640
esp_ble_mesh_gen_def_trans_time_status_cb_t
(C++ struct), 635
esp_ble_mesh_gen_def_trans_time_set_t::trans_time
(C++ union), 635
esp_ble_mesh_gen_client_status_status_cb_t::admin_property_status
(C++ struct), 635
esp_ble_mesh_gen_client_status_status_cb_t::admin_properties_status
(C++ struct), 635
esp_ble_mesh_gen_client_status_status_cb_t::property_ids
(C++ struct), 635
esp_ble_mesh_gen_client_properties_get_t::property_id
(C++ struct), 635
esp_ble_mesh_gen_client_properties_get_t
(C++ struct), 635
esp_ble_mesh_gen_client_prop_srv_t::rsp_ctrl
(C++ struct), 635
esp_ble_mesh_gen_client_prop_srv_t::property_ids
(C++ struct), 635
esp_ble_mesh_gen_client_prop_srv_t::model
(C++ struct), 635
esp_ble_mesh_gen_level_state_t::last_level
(C++ struct), 635
esp_ble_mesh_gen_level_state_t::last_delta
(C++ struct), 635
esp_ble_mesh_gen_level_srv_t::tt_delta_level
(C++ struct), 635
esp_ble_mesh_gen_level_srv_t::transition
(C++ struct), 635
esp_ble_mesh_gen_level_srv_t::state
(C++ struct), 635
esp_ble_mesh_gen_level_srv_t::rsp_ctrl
(C++ struct), 635
esp_ble_mesh_gen_level_srv_t::model
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::trans_time
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::tid
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::op_en
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::level
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::delay
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::trans_time
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::tid
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::level
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::delay
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::trans_time
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::tid
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::level
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::delay
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::trans_time
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::tid
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::level
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::delay
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::trans_time
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::tid
(C++ struct), 635
esp_ble_mesh_gen_level_set_t::level
(C++ struct), 635
esp_ble_mesh_gen_def_trans_time_set_t::esp_ble_mesh_gen_def_trans_time_set_t::last
(C++ member), 650
esp_ble_mesh_gen_def_trans_time_set_t::esp_ble_mesh_gen_def_trans_time_set_t::model
(C++ member), 650
esp_ble_mesh_gen_def_trans_time_set_t::esp_ble_mesh_gen_def_trans_time_set_t::rsp_ctrl
(C++ member), 650
esp_ble_mesh_gen_def_trans_time_set_t::esp_ble_mesh_gen_def_trans_time_set_t::state
(C++ member), 650
esp_ble_mesh_gen_def_trans_time_set_t::esp_ble_mesh_gen_def_trans_time_set_t::transition
(C++ member), 650
esp_ble_mesh_gen_def_trans_time_set_t::esp_ble_mesh_gen_def_trans_time_set_t::tt_delta_level
(C++ member), 650
esp_ble_mesh_gen_def_trans_time_set_t::esp_ble_mesh_gen_def_trans_time_set_t::last_delta
(C++ member), 650
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::last
(C++ member), 651
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::model
(C++ member), 651
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::rsp_ctrl
(C++ member), 651
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::state
(C++ member), 651
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::transition
(C++ member), 651
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::tt_delta_level
(C++ member), 651
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::last_delta
(C++ member), 651
esp_ble_mesh_gen_def_trans_time_srv_t::esp_ble_mesh_gen_def_trans_time_srv_t::last_level
(C++ member), 651
esp_ble_mesh_gen_level_state_t::level  esp_ble_mesh_gen_location_setup_srv_t
  (C++ member), 650
esp_ble_mesh_gen_level_state_t::model  esp_ble_mesh_gen_location_setup_srv_t::model
  (C++ member), 655
esp_ble_mesh_gen_level_state_t::position  esp_ble_mesh_gen_location_setup_srv_t::rsp_ctrl
  (C++ member), 655
esp_ble_mesh_gen_level_state_t::target  esp_ble_mesh_gen_location_setup_srv_t::state
  (C++ member), 655
esp_ble_mesh_gen_level_status_cb_t  esp_ble_mesh_gen_location_srv_t  (C++
  struct), 654
esp_ble_mesh_gen_level_status_cb_t::op  esp_ble_mesh_gen_location_srv_t::model
  (C++ member), 655
esp_ble_mesh_gen_level_status_cb_t::positive  esp_ble_mesh_gen_location_srv_t::rsp_ctrl
  (C++ member), 655
esp_ble_mesh_gen_level_status_cb_t::target  esp_ble_mesh_gen_location_state_t
  (C++ member), 655
esp_ble_mesh_gen_loc_global_set_t  esp_ble_mesh_gen_location_state_t::floor_number
  (C++ struct), 644
esp_ble_mesh_gen_loc_global_set_t::global_altitude  esp_ble_mesh_gen_location_state_t::global_latitude
  (C++ member), 644
est_pble_mesh_gen_location_state_t::local_east
  esp BLE_MESH_GEN_LOCATION_STATE_T::local_east
  (C++ struct), 646
est_pble_mesh_gen_location_state_t::local_north
  esp BLE_MESH_GEN_LOCATION_STATE_T::local_north
  (C++ struct), 646
est_pble_mesh_gen_location_state_t::local_altitude
  esp BLE_MESH GEN_LOCATION STATE T::local_altitude
  (C++ member), 646
est_pble_mesh_gen_location_state_t::local_east
  esp BLE_MESH GEN_LOCATION STATE T::local_east
  (C++ member), 646
est_pble_mesh_gen_location_state_t::local_north
  esp BLE_MESH GEN_LOCATION STATE T::local_north
  (C++ member), 646
est_pble_mesh_gen_location_state_t::uncertainty
  esp BLE_MESH GEN_LOCATION STATE T::uncertainty
  (C++ member), 646
est_pble_mesh_gen_location_set_t  esp BLE_MESH GEN_MANU_PROP_ACCESS_T
  (C++ struct), 654
esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ member), 657
est_pble_mesh_gen_location_set_t::floor_number
  esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ member), 657
est_pble_mesh_gen_location_set_t::local_east
  esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ member), 657
est_pble_mesh_gen_location_set_t::local_north
  esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ member), 657
est_pble_mesh_gen_location_set_t::local_east
  esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ struct), 656
est_pble_mesh_gen_location_set_t::local_north
  esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ struct), 656
est_pble_mesh_gen_location_set_t::local_east
  esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ member), 656
est_pble_mesh_gen_location_set_t::local_north
  esp BLE_MESH GEN MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ member), 656
est_pble_mesh_gen_location_status_cb_t  esp BLE_MESH GEN_MANU_PROP_ACCESS_T::ESP_BLE_MESH
  (C++ struct), 656
est_pble_mesh_gen_location_status_cb_t::ESP_BLE_MESH
  (C++ member), 656
est_pble_mesh_gen_location_status_cb_t::esp BLE_MESH
  (C++ member), 656
est_pble_mesh_gen_location_status_cb_t::ESP_BLE_MESH
  (C++ member), 656
est_pble_mesh_gen_location_status_cb_t::ESP_BLE_MESH
  (C++ member), 656
Esp ble mesh gen onoff status cb t (C++ struct), 643
Esp ble mesh gen manufacturer property esp ble mesh gen onoff status cb t::op en (C++ member), 644
Esp ble mesh gen manufacturer property esp ble mesh gen onoff status cb t::present onoff (C++ member), 644
Esp ble mesh gen manufacturer property esp ble mesh gen onoff status cb t::remain time (C++ member), 644
Esp ble mesh gen onpowerup status cb t (C++ struct), 640
Esp ble mesh gen onpowerup state t (C++ struct), 640
Esp ble mesh gen onpowerup state t::onpowerup (C++ member), 641
Esp ble mesh gen move set t (C++ struct), esp ble mesh gen onpowerup state t::onpowerup (C++ member), 651
Esp ble mesh gen move set t::delay esp ble mesh gen onpowerup status cb t (C++ struct), 645
Esp ble mesh gen move set t::delta level esp ble mesh gen onpowerup status cb t::onpowerup (C++ member), 654
Esp ble mesh gen move set t::op en esp ble mesh gen power default set t (C++ struct), 645
Esp ble mesh gen move set t::tid (C++ member), esp ble mesh gen power default set t::power (C++ member), 654
Esp ble mesh gen move set t::trans time esp ble mesh gen power default status cb t (C++ struct), 645
Esp ble mesh gen onoff set t (C++ struct), esp ble mesh gen power default status cb t::power (C++ member), 654
Esp ble mesh gen onoff set t::delay esp ble mesh gen power last status cb t (C++ struct), 645
Esp ble mesh gen onoff set t::onoff esp ble mesh gen power last status cb t::power (C++ member), 654
Esp ble mesh gen onoff set t::op en esp ble mesh gen power level set t (C++ struct), 641
Esp ble mesh gen onoff set t::tid esp ble mesh gen power level set t::delay (C++ member), 641
Esp ble mesh gen onoff set t::trans time esp ble mesh gen power level set t::op en (C++ member), 641
Esp ble mesh gen onoff srv t (C++ struct), esp ble mesh gen power level set t::power (C++ member), 641
Esp ble mesh gen onoff srv t::last esp ble mesh gen power level set t::tid (C++ member), 641
Esp ble mesh gen onoff srv t::model esp ble mesh gen power level set t::trans time (C++ member), 641
Esp ble mesh gen onoff srv t::rsp ctrl esp ble mesh gen power level setup srv t (C++ struct), 653
Esp ble mesh gen onoff srv t::state esp ble mesh gen power level setup srv t::model (C++ member), 653
Esp ble mesh gen onoff srv t::transition esp ble mesh gen power level setup srv t::rsp ctrl (C++ member), 653
Esp ble mesh gen onoff state t (C++ struct), 649
Esp ble mesh gen onoff state t::onoff esp ble mesh gen power level srv t (C++ struct), 652
Esp ble mesh gen onoff state t::target esp ble mesh gen power level srv t::last (C++ member), 653
esp_ble_mesh_generic_client_set_state_t::power_set
(\texttt{C++ member}), 649

esp_ble_mesh_generic_client_set_state_t::power_range_set
(\texttt{C++ member}), 649

esp_ble_mesh_generic_client_set_state_t::power_level_set
(\texttt{C++ member}), 649

esp_ble_mesh_generic_client_set_state_t::power_default_set
(\texttt{C++ member}), 649

esp_ble_mesh_generic_client_set_state_t::manufacturer_property_set
(\texttt{C++ member}), 655

esp_ble_mesh_generic_client_set_state_t::level_set
(\texttt{C++ member}), 655

esp_ble_mesh_generic_client_set_state_t::delta_set
(\texttt{C++ member}), 655

esp_ble_mesh_generic_client_set_state_t::def_trans_time_set
(\texttt{C++ member}), 655

esp_ble_mesh_generic_client_set_state_t::admin_property_set
(\texttt{C++ member}), 655

esp_ble_mesh_generic_client_set_state_t::user_property_set
(\texttt{C++ member}), 655

esp_ble_mesh_generic_client_t (\texttt{C++ function}), 671

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_cb_event_t::ESP_BLE_MESH_GENERIC_CLIENT_TIMEOUT_EVT
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_cb_event_t::ESP_BLE_MESH_GENERIC_CLIENT_SET_STATE_EVT
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_cb_event_t::ESP_BLE_MESH_GENERIC_CLIENT_PUBLISH_EVT
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_cb_value_t::state_change
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_cb_value_t::set
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_cb_value_t::get
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_get_set_msg_t:::ESP_BLE_MESH_GENERIC_SERVER_RECV_SET_MSG_EVT
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_get_msg_t::user_property
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_get_msg_t::manu_property
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_get_msg_t::client_properties
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_get_msg_t::admin_property
(\texttt{C++ function}), 672

esp_ble_mesh_generic_client_get_state esp_ble_mesh_generic_server_set_msg_t
(\texttt{C++ member}), 634

esp_ble_mesh_generic_set_msg_opcode_t
(\texttt{C++ member}), 634

esp_ble_mesh_generic_property_t::val
(\texttt{C++ member}), 634

esp_ble_mesh_generic_property_t::user_access
(\texttt{C++ member}), 634

esp_ble_mesh_generic_property_t::manu_access
(\texttt{C++ member}), 634

esp_ble_mesh_generic_property_t::admin_access
(\texttt{C++ member}), 634

esp_ble_mesh_generic_message_opcode_t
esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::trans_time
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_set_t::tid
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_set_t::hsl_saturation
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_set_t::delay
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_status_cb_t
  esp_ble_mesh_light_hsl_saturation_status_cb_t::op_en
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_status_cb_t::present_saturation
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_status_cb_t::target_saturation
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_status_cb_t::remain_time
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_status_cb_t::target_hue
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_status_cb_t::remain_time
  (C++ member), 740
  esp_ble_mesh_light_hsl_saturation_status_cb_t::present_saturation
  (C++ member), 740
  esp_ble_mesh_light_hsl_range_set_t
  esp_ble_mesh_light_hsl_range_set_t::saturation_range_max
  (C++ member), 732
  esp_ble_mesh_light_hsl_range_set_t::hue_range_max
  (C++ member), 732
  esp_ble_mesh_light_hsl_range_set_t::hue_range_min
  (C++ member), 732
  esp_ble_mesh_light_hsl_range_set_t
  esp_ble_mesh_light_hsl_range_status_cb_t::status_code
  (C++ member), 732
  esp_ble_mesh_light_hsl_range_status_cb_t::saturation_range_min
  (C++ member), 732
  esp_ble_mesh_light_hsl_range_status_cb_t::saturation_range_max
  (C++ member), 732
  esp_ble_mesh_light_hsl_range_status_cb_t::hue_range_min
  (C++ member), 732
  esp_ble_mesh_light_hsl_range_status_cb_t::hue_range_max
  (C++ member), 732
  esp_ble_mesh_light_hsl_hue_status_cb_t
  esp_ble_mesh_light_hsl_hue_status_cb_t::target_hue
  (C++ member), 732
  esp_ble_mesh_light_hsl_hue_status_cb_t::remain_time
  (C++ member), 732
  esp_ble_mesh_light_hsl_hue_status_cb_t::present_hue
  (C++ member), 732
  esp_ble_mesh_light_hsl_hue_status_cb_t::op_en
  (C++ member), 732
  esp_ble_mesh_light_hsl_set_t
  esp_ble_mesh_light_hsl_set_t::trans_time
  (C++ member), 732
  esp_ble_mesh_light_hsl_set_t::tid
  (C++ member), 732
  esp_ble_mesh_light_hsl_set_t::op_en
  (C++ member), 732
  esp_ble_mesh_light_hsl_set_t::hsl_saturation
  (C++ member), 732
  esp_ble_mesh_light_hsl_set_t::hsl_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_set_t::hsl_hue
  (C++ member), 732
  esp_ble_mesh_light_hsl_set_t::delay
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t
  esp_ble_mesh_light_hsl_setup_srv_t::state
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::model
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::last
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::mode
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::last
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::model
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::model
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::state
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::state
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::state
  (C++ member), 732
  esp_ble_mesh_light_hsl_setup_srv_t::transition
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_hue
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_saturation
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp_ble_mesh_light_hsl_saturation_set_t
  esp_ble_mesh_light_hsl_saturation_set_t::tt_delta_lightness
  (C++ member), 732
  esp BLE Mesh Light HSL Saturation Status Callback
(C++ struct), 743
esp_ble_mesh_light_lc_property_state_t::target_light_onoff (C++ member), 753
esp_ble_mesh_light_lc_property_state_t::ambient_luxlevel (C++ member), 753
esp_ble_mesh_light_lc_property_state_t::light_onoff (C++ member), 753
esp_ble_mesh_light_lc_property_state_t::linear_output (C++ member), 753
esp_ble_mesh_light_lc_property_state_t::mode (C++ member), 753
esp_ble_mesh_light_lc_property_state_t::occupancy (C++ member), 753
esp_ble_mesh_light_lc_property_state_t::occupancy_mode (C++ member), 753
esp_ble_mesh_light_lc_property_status.cb_t (C++ struct), 729
dobbleble_mesh_light_lightness_default_set_t (C++ struct), 729
dobbleble_mesh_light_lightness_default_status_cb_t (C++ struct), 729
dobbleble_mesh_light_lightness_last_status CB_T (C++ struct), 737
dobbleble_mesh_light_lightness_linear_set_t (C++ struct), 729
dobbleble_mesh_light_lightness_linear_last_status CB_T (C++ struct), 737
dobbleble_mesh_light_lightness_linear_set_t (C++ struct), 729
dobbleble_mesh_light_lightness_linear_set_t::delay (C++ member), 729
dobbleble_mesh_light_lightness_linear_set_t::lightn (C++ member), 729
dobbleble_mesh_light_lightness_linear_set_t::tid (C++ member), 729
dobbleble_mesh_light_lightness_linear_set_t::trans (C++ member), 729
dobbleble_mesh_light_lightness_linear_set_t::transition (C++ struct), 736
dobbleble_mesh_light_lightness_linear_status cb t (C++ member), 729
debleble_mesh_light_lightness_linear_status cb t (C++ member), 729
debleble_mesh_light_lightness_linear_status cb t (C++ member), 729
debleble_mesh_light_lightness_linear_status cb t (C++ member), 729
debleble_mesh_light_lightness_linear_status cb t (C++ member), 729
EsperrySystems
Submit Document Feedback

Esperry Systems
2581 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
索引

(C++ member), 730
esp_ble_mesh_lightness_range_status_cb_t::status_code (C++ member), 745

(C++ struct), 737
esp_ble_mesh_lightness_range_state_t::lightness_range_max (C++ member), 744

(C++ member), 737
esp_ble_mesh_lightness_range_state_t::status_code (C++ member), 744

(C++ member), 737
esp_ble_mesh_lightness_range_state_t::target_lightness (C++ member), 744

(C++ member), 737
desp_ble_mesh_lightness_setup_srv_t::state (C++ member), 744

(C++ struct), 728
desp_ble_mesh_lightness_lightness_srv_t::state (C++ member), 744

(C++ member), 729
desp_ble_mesh_lightness_lightness_srv_t::rsp_ctrl (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_srv_t::model (C++ member), 744

(C++ member), 728
desp_ble_mesh_lightness_lightness_srv_t::linear_transition (C++ member), 744

(C++ struct), 728
desp_ble_mesh_lightness_lightness_set_t::trans_time (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_set_t::tid (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_set_t::op_en (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_set_t::lightness (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_set_t::delay (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::model (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::lightness_range_status_cb_t (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::lightness_range_status_cb_t (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::xyl_y_range_status_cb_t (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::xyl_x_range_status_cb_t (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::xyl_y_range_set_t (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::xyl_x_range_set_t (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::xyl_y (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::xyl_x (C++ member), 744

(C++ struct), 729
desp_ble_mesh_lightness_lightness_setup_srv_t::target_lightness (C++ member), 744

(C++ type), 561
desp_ble_mesh_lightness_setup_msg::message_opcode_t (C++ member), 744

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_default_set_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_range_set_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_y (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_x (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_status_cb_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_default_status_cb_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_y_range_set_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_x_range_set_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_y_range_status_cb_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_x_range_status_cb_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_y (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_x (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_y_range_status_cb_t (C++ member), 745

(C++ struct), 745
desp_ble_mesh_lightness_setup_srv_t::esp_ble_mesh_light_xyl_x_range_status_cb_t (C++ member), 745
Espresif Systems

Submit Document Feedback
ESP_BLE_MESH_MODEL_OP_ATTENTION_SET_UNACK (C macro), 550
ESP_BLE_MESH_MODEL_OP_ATTENTION_STATUS (C macro), 550
ESP_BLE_MESH_MODEL_OP_BEACON_GET (C macro), 544
ESP_BLE_MESH_MODEL_OP_BEACON_SET (C macro), 545
ESP_BLE_MESH_MODEL_OP_BEACON_STATUS (C macro), 547
ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET (C macro), 544
ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_STATUS (C macro), 547
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_GET (C macro), 544
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET (C macro), 545
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_STATUS (C macro), 547
ESP_BLE_MESH_MODEL_OP_END (C macro), 541
ESP_BLE_MESH_MODEL_OP_FRIEND_GET (C macro), 545
ESP_BLE_MESH_MODEL_OP_FRIEND_SET (C macro), 546
ESP_BLE_MESH_MODEL_OP_FRIEND_STATUS (C macro), 548
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_GET (C macro), 544
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET (C macro), 545
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_STATUS (C macro), 547
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_STATUS (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET_UNACK (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_GET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_GET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_SET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_STATUS (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_STATUS (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_GET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET_UNACK (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET_UNACK (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_GET (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET_UNACK (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_GET (C macro), 551
ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_STATUS (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_GET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_STATUS (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_STATUS (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_STATUS (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET_UNACK (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_GET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET (C macro), 552
ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET_UNACK (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_BLE_MESH_MODEL_OP_GEN_MOVE_SET_UNACK (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_BLE_MESH_MODEL_OP_GEN_ONOFF_STATUS (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_BLE_MESH_MODEL_OP_GEN_ONOFF_SET (C macro), 550
ESP_BLE_MESH_MODEL_OP_GEN_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET (C macro), 551
ESP_BLE_MESH_MODEL_OP_GEN_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_GET (C macro), 551
ESP_BLE_MESH_MODEL_OP_GEN_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET (C macro), 551
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_O Target
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_STATUS
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET_UNACK
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_GET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET_UNACK
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET_UNACK
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_UNBIND
ESP_BLE_MESH_MODEL_OP_RELAY_UNBIND
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_OP_RELAY_SET
ESP_BLE_MESH_MODEL_OP_RELAY_GET
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS
ESP_BLE_MESH_MODEL_O
esp_ble_mesh_model_op_t::opcode

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET_UNACK (C++ member), 528
ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_STATUS (C macro), 555
ESP_BLE_MESH_MODEL_OP_SCHEDULER_GET (C macro), 555

ESP_BLE_MESH_MODEL_OP_SCHEDULER_STATUS (C macro), 555
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_STATUS (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK (C macro), 554

ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_GET (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_STATUS (C macro), 553

ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET (C macro), 553

ESP_BLE_MESH_MODEL_OP_SENSOR_GET (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_GET (C macro), 553
ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_STATUS (C macro), 553

ESP_BLE_MESH_MODEL_OP_TIME_ZONE_GET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_STATUS (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_SET (C macro), 554

ESP_BLE_MESH_MODEL_OP_TIME_ROLE_GET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_SET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_STATUS (C macro), 554

ESP_BLE_MESH_MODEL_OP_TIME_GET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_SET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_STATUS (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_GET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_SET (C macro), 554

ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_GET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_SET (C macro), 554
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_STATUS (C macro), 554

esp_ble_mesh_model_op_t::app_idx (C++ member), 527
esp_ble_mesh_model_op_t::count (C++ member), 527
esp_ble_mesh_model_op_t::cred (C++ member), 527
esp_ble_mesh_model_op_t::dev_role (C++ member), 528
esp_ble_mesh_model_op_t::fast_period (C++ member), 527
esp_ble_mesh_model_op_t::model (C++ member), 527
esp_ble_mesh_model_op_t::msg (C++ member), 527
esp_ble_mesh_model_op_t::period (C++ member), 527
esp_ble_mesh_model_op_t::period_div (C++ member), 527
esp_ble_mesh_model_op_t::publish_addr (C++ member), 528
(C++ member), 527
esp_ble_mesh_model_pub_t::retransmit (C++ member), 527
esp_ble_mesh_model_pub_t::send_rel (C++ member), 527
esp_ble_mesh_model_pub_t::timer (C++ member), 528
esp_ble_mesh_model_pub_t::ttl (C++ member), 527
esp_ble_mesh_model_pub_t::update (C++ member), 528
esp_ble_mesh_model_publish (C++ function), 576
ESP_BLE_MESH_MODEL_SCENE_CLI (C macro), 715
ESP_BLE_MESH_MODEL_SCENE_SETUP_SRV (C macro), 717
ESP_BLE_MESH_MODEL_SCENE_SRV (C macro), 716
ESP_BLE_MESH_MODEL_SCHEDULER_CLI (C macro), 715
ESP_BLE_MESH_MODEL_SCHEDULER_SETUP_SRV (C macro), 717
ESP_BLE_MESH_MODEL_SCHEDULER_SRV (C macro), 717
ESP_BLE_MESH_MODEL_SENSOR_CLI (C macro), 687
ESP_BLE_MESH_MODEL_SENSOR_SETUP_SRV (C macro), 688
ESP_BLE_MESH_MODEL_SENSOR_SRV (C macro), 688
ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MAX (C macro), 538
ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MIN (C macro), 538
ESP_BLE_MESH_MODEL_STATUS_SUCCESS (C macro), 559
esp_ble_mesh_model_status_t (C++ type), 561
esp_ble_mesh_model_subscribe_group_addr (C++ function), 572
esp_ble_mesh_model_t (C++ type), 560
ESP_BLE_MESH_MODEL_TIME_CLI (C macro), 715
ESP_BLE_MESH_MODEL_TIME_SETUP_SRV (C macro), 716
ESP_BLE_MESH_MODEL_TIME_SRV (C macro), 716
esp_ble_mesh_model_unsubscribe_group_addr (C++ function), 573
esp_ble_mesh_msg_ctx_t (C++ struct), 529
esp_ble_mesh_msg_ctx_t::addr (C++ member), 530
esp_ble_mesh_msg_ctx_t::app_idx (C++ member), 530
esp_ble_mesh_msg_ctx_t::model (C++ member), 530
esp_ble_mesh_msg_ctx_t::net_idx (C++ member), 530
esp_ble_mesh_msgctxt::net_idx (C++ member), 530
esp_ble_mesh_msgctxt::recv_dst (C++ member), 530
esp_ble_mesh_msgctxt::recv_op (C++ member), 530
esp_ble_mesh_msgctxt::recv_rssi (C++ member), 530
esp_ble_mesh_msgctxt::recv_ttl (C++ member), 530
esp_ble_mesh_msgctxt::send_rel (C++ member), 530
esp_ble_mesh_msgctxt::send_ttl (C++ member), 530
esp_ble_mesh_msg_ctxt_t::srv_send (C++ member), 530
ESP_BLE_MESH_NET_PRIMARY (C macro), 538
ESP_BLE_MESH_NO_FAULT (C macro), 629
ESP_BLE_MESH_NO_LOAD_ERROR (C macro), 629
ESP_BLE_MESH_NO_LOAD_WARNING (C macro), 629
esp_ble_mesh_node_add_local_app_key (C++ function), 573
esp_ble_mesh_node_add_local_net_key (C++ function), 573
esp_ble_mesh_node_bind_app_key_to_local_model (C++ function), 573
esp_ble_mesh_node_get_local_app_key (C++ function), 573
esp_ble_mesh_node_get_local_net_key (C++ function), 573
ESP_BLE_MESH_NODE_IDENTITY_NOT_SUPPORTED (C macro), 538
ESP_BLE_MESH_NODE_IDENTITY_RUNNING (C macro), 538
ESP_BLE_MESH_NODE_IDENTITY_STOPPED (C macro), 538
esp_ble_mesh_node_input_number (C++ function), 584
esp_ble_mesh_node_input_string (C++ function), 584
esp_ble_mesh_node_is_provisioned (C++ function), 583
esp_ble_mesh_node_local_reset (C++ function), 577
ESP_BLE_MESH_NODE_NAME_MAX_LEN (C macro), 537
esp_ble_mesh_node_prov_disable (C++ function), 583
esp_ble_mesh_node_prov_enable (C++ function), 583
esp_ble_mesh_node_set_oob_pub_key (C++ function), 583
esp_ble_mesh_node_t (C++ struct), 532
esp_ble_mesh_node_t::addr (C++ member), 532
esp_ble_mesh_node_t::addr_type (C++ member), 532
esp_ble_mesh_node_t::comp_data (C++ member), 532

Espersif Systems 2594 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
(C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_RESET_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_INPUT_NUMBER_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_NODE_PROV_INPUT_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_INDEX_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROVISIONER_ENABLE_HEARTBEAT_RECV_COMP_EVT  (C++ enumerator), 564  (C++ enumerator), 567
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_SERVER_DISCONNECTED_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_SERVER_CONNECTED_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT
esp_ble_mesh_prov_cb_event_t::ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT
esp_ble_mesh_prov_cb_param_t::ble_mesh_event_param::ESP_BLE_MESH_INPUT_EVT
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_evt_param::size
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_evt_param::action
esp_ble_mesh_prov_cb_param_t::ble_mesh_model_unsub_group_addr_comp_param::element_addr
esp_ble_mesh_prov_cb_param_t::ble_mesh_model_sub_group_addr_comp_param::group_addr
esp_ble_mesh_prov_cb_param_t::ble_mesh_model_sub_group_addr_comp_param::err_code
esp_ble_mesh_prov_cb_param_t::ble_mesh_model_sub_group_addr_comp_param::company_id
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_poll_comp_param
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_terminate_param::friend_addr
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_establish_param::friend_addr
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_enable_comp_param
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_disable_comp_param::err_code
esp_ble_mesh_prov_cb_param_t::ble_mesh_link_open_evt_param::bearer
esp_ble_mesh_prov_cb_param_t::ble_mesh_link_close_evt_param
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_string_comp_param
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_number_comp_param
(C++ struct), 520
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_set_unprov_dev_name_comp::err_code
(C++ member), 501
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_set_oob_pub_key_comp_param::err_code
(C++ member), 501
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_set_fast_prov_info_comp_param::status_unicast
(C++ member), 500
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_set_fast_prov_info_comp_param::status_net_idx
(C++ member), 500
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_set_fast_prov_info_comp_param::status_match
(C++ member), 500
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_server_disconnected_param::conn_handle
(C++ member), 501
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_server_connected_param::conn_handle
(C++ member), 501

-espressifsystems- 2600  
Submit Document Feedback
(C++ function), 579
esp_ble_mesh_provisioner_update_local_net_key (C macro), 538
esp_ble_mesh_scene_delete_t (C++ struct), 698
esp_ble_mesh_scene_delete_t::scene_number (C++ member), 699
ESP_BLE_MESH_SCENE_NOT_FOUND (C macro), 719
ESP_BLE_MESH_SCENE_NUMBER_LEN (C macro), 718
esp_ble_mesh_scene_recall_t (C++ struct), 698
esp_ble_mesh_scene_recall_t::delay (C++ member), 698
esp_ble_mesh_scene_recall_t::op_en (C++ member), 698
esp_ble_mesh_scene_recall_t::scene_number (C++ member), 698
esp_ble_mesh_scene_recall_t::trans_time (C++ member), 698
ESP_BLE_MESH_SCENE_REG_FULL (C macro), 719
esp_ble_mesh_scene_register_status_cb_t (C++ struct), 701
esp_ble_mesh_scene_register_status_cb_t::current_scene (C++ member), 702
esp_ble_mesh_scene_register_status_cb_t::scenes (C++ member), 702
esp_ble_mesh_scene_register_status_cb_t::status_code (C++ member), 702
esp_ble_mesh_scene_register_t (C++ struct), 704
 esp_ble_mesh_scene_register_t::scene_number (C++ member), 705
esp_ble_mesh_scene_register_t::scene_type (C++ member), 705
esp_ble_mesh_scene_register_t::scene_value (C++ member), 705
esp_ble_mesh_scene_delete_t (C++ struct), 706
esp_ble_mesh_scene_setup_srv_t (C++ struct), 706
esp_ble_mesh_scene_setup_srv_t::model (C++ member), 707
esp_ble_mesh_scene_setup_srv_t::state (C++ member), 707
esp_ble_mesh_scene_setup_srv_t::model (C++ member), 707
esp_ble_mesh_scene_setup_srv_t::state (C++ member), 707
ESP_BLE_MESH_RELAY_NOT_SUPPORTED (C macro), 538
ESP_BLE_MESH_RELAY_ENABLED (C macro), 538
esp_ble_mesh_schedule_register_t::day_of_week
esp_ble_mesh_schedule_register_t::day
 esp_ble_mesh_schedule_register_t::op_en
 esp_ble_mesh_schedule_register_t::year
 esp_ble_mesh_schedule_register_t::trans_time
 esp_ble_mesh_schedule_register_t::scene_number
 esp_ble_mesh_schedule_register_t::second
 esp_ble_mesh_schedule_register_t::month
 esp_ble_mesh_schedule_register_t::status_code
 esp_ble_mesh_scenes_state_t::target_scene
 esp_ble_mesh_scenes_state_t::status_code
 esp_ble_mesh_scenes_state_t::scenes
 esp_ble_mesh_scenes_state_t::scene_count
 esp_ble_mesh_scenes_state_t::in_progress
 esp_ble_mesh_scenes_state_t::current_scene
 esp_ble_mesh_scenes_state_t
 esp_ble_mesh_scenes_state_t::in_use
 esp_ble_mesh_scenes_state_t::minute
 esp_ble_mesh_scenes_state_t::month
 esp_ble_mesh_scenes_state_t::year
 ESP_BLE_MESH_SCHEDULE_SCENE_NO_SCENE
 ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_HOUR
 ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_SEC
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_SEC
 ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_HOUR
 ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_SEC
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_SEC
 ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_HOUR
 ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_MIN
 ESP_BLE_MESH_SCHEDULE_SCENE_NO_SCENE
 ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_HOUR
 ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_SEC
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_MIN
 ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_SEC
 ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_HOUR
 ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_MIN
 ESP_BLE_MESH_SCHEDULE_SCENE_NO_SCENE
esp_ble_mesh_scheduler_act_set_t::trans_time (C++ member), 630
esp_ble_mesh_scheduler_act_set_t::year (C++ member), 699
esp_ble_mesh_scheduler_set_srv_t (C++ struct), 708
esp_ble_mesh_sensor_client_cb_event_t::ESP_BLE_MESH_SENSOR_CLIENT_TIMEOUT_EVT (C++ event), 703
esp_ble_mesh_sensor_client_cb_event_t::ESP_BLE_MESH_SENSOR_CLIENT_SET_STATE_EVT (C++ event), 703
esp_ble_mesh_sensor_client_cb_event_t::ESP_BLE_MESH_SENSOR_CLIENT_GET_STATE_EVT (C++ event), 703
esp_ble_mesh_sensor_client_cb_event_t::ESP_BLE_MESH_SENSOR_CLIENT_EVT_MAX (C++ event), 703
esp_ble_mesh_sensor_cadence_t::trigger_type (C++ member), 682
esp_ble_mesh_sensor_cadence_t::trigger_delta_up (C++ member), 682
esp_ble_mesh_sensor_cadence_t::trigger_delta_down (C++ member), 682
esp_ble_mesh_sensor_cadence_t::min_interval (C++ member), 682
esp_ble_mesh_sensor_cadence_get_t::sensor_cadence_value (C++ member), 676
esp_ble_mesh_sensor_cadence_get_t::property_id (C++ member), 676
esp_ble_mesh_sensor_cadence_set_t::status_trigger_type (C++ member), 679
esp_ble_mesh_sensor_cadence_set_t::status_trigger_delta_up (C++ member), 679
esp_ble_mesh_sensor_cadence_set_t::status_trigger_delta_down (C++ member), 679
esp_ble_mesh_sensor_cadence_set_t::status_min_interval (C++ member), 679
esp_ble_mesh_sensor_cadence_get_t::property_id (C++ member), 676
esp_ble_mesh_sensor_cadence_set_t::fast_cadence_period_divisor (C++ member), 677
esp_ble_mesh_sensor_cadence_set_t::fast_cadence_low (C++ member), 677
esp_ble_mesh_sensor_cadence_set_t::fast_cadence_high (C++ member), 677
esp_ble_mesh_sensor_cadence_set_t::min_interval (C++ member), 677
esp_ble_mesh_sensor_cadence_set_t (C++ struct), 681
esp_ble_mesh_sensor_cadence_status_cb_t::sensor_cadence_value (C++ member), 679
esp_ble_mesh_sensor_cadence_status_cb_t::property_id (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_high (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_low (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_period_divisor (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_low (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t (C++ struct), 679
esp_ble_mesh_sensor_cadence_status_cb_t::status_min_interval (C++ member), 679
esp_ble_mesh_sensor_cadence_status_cb_t::status_min_interval (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t::property_id (C++ member), 676
esp_ble_mesh_sensor_cadence_cadence_status_cb_t (C++ struct), 679
esp_ble_mesh_sensor_cadence_cadence_status_cb_t (C++ struct), 677
esp_ble_mesh_sensor_cadence_cadence_set_t (C++ struct), 679
esp_ble_mesh_sensor_cadence_cadence_get_t (C++ struct), 677
esp_ble_mesh_sensor_cadence_cadence_set_t (C++ struct), 677
esp_ble_mesh_sensor_cadence_cadence_get_t (C++ struct), 677
esp_ble_mesh_sensor_cadence_get_t (C++ struct), 676
esp_ble_mesh_sensor_cadence_set_t (C++ struct), 676
esp_ble_mesh_sensor_cadence_status_cb_t::sensor_cadence_value (C++ member), 679
esp_ble_mesh_sensor_cadence_status_cb_t::property_id (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_high (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_low (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_period_divisor (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_low (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t (C++ struct), 679
esp_ble_mesh_sensor_cadence_status_cb_t::status_min_interval (C++ member), 679
esp_ble_mesh_sensor_cadence_status_cb_t::status_min_interval (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t::property_id (C++ member), 676
esp_ble_mesh_sensor_cadence_cadence_status_cb_t (C++ struct), 679
esp_ble_mesh_sensor_cadence_cadence_status_cb_t (C++ struct), 677
esp_ble_mesh_sensor_cadence_cadence_set_t (C++ struct), 679
esp_ble_mesh_sensor_cadence_cadence_get_t (C++ struct), 677
esp_ble_mesh_sensor_cadence_get_t (C++ struct), 676
esp_ble_mesh_sensor_cadence_set_t (C++ struct), 676
esp_ble_mesh_sensor_cadence_status_cb_t::sensor_cadence_value (C++ member), 679
esp_ble_mesh_sensor_cadence_status_cb_t::property_id (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_high (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_low (C++ member), 676
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_period_divisor (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t::fast_cadence_low (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t (C++ struct), 679
esp_ble_mesh_sensor_cadence_status_cb_t::status_min_interval (C++ member), 679
esp_ble_mesh_sensor_cadence_status_cb_t::status_min_interval (C++ member), 677
esp_ble_mesh_sensor_cadence_status_cb_t::property_id (C++ member), 676
ESP_BLE_MESH_SDÚ_MAX_LEN (C macro), 537
ESP_BLE_MESH_SELF_TEST_WARNING (C macro), 630
ESP_BLE_MESH_SELF_TEST_ERROR (C macro), 630
ESP_BLE_MESH_SDÚ_MAX_LEN (C macro), 537
ESP_BLE_MESH_SELF_TEST_WARNING (C macro), 630
Espresif Systems

2607 Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
Espresif Systems 2610

Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
esp_ble_mesh_tai_utc_delta_status_cb_t::tai_utc_delta_curr
esp_ble_mesh_tai_utc_delta_status_cb_t::tai_utc_delta_change
esp_ble_mesh_tai_utc_delta_status_cb_t::padding_2

esp_ble_mesh_time_role_set_t::time_role
esp_ble_mesh_time_role_set_t::ESP_BLE_MESH_TIME_RELAY
esp_ble_mesh_time_role_set_t::ESP_BLE_MESH_TIME_NONE
esp_ble_mesh_time_role_set_t::ESP_BLE_MESH_TIME_CLINET

esp_ble_mesh_time_scene_client_set_state_t::time_zone_set
esp_ble_mesh_time_scene_client_set_state_t::time_role_set
esp_ble_mesh_time_scene_client_set_state_t::tai_utc_delta_set
esp_ble_mesh_time_scene_client_set_state_t::scheduler_act_set
esp_ble_mesh_time_scene_client_set_state_t::scene_store

esp_ble_mesh_time_scene_server_cb_value_t
esp_ble_mesh_time_scene_server_cb_t
esp_ble_mesh_time_scene_server_cb_param_t::value
esp_ble_mesh_time_scene_server_cb_param_t::model
esp_ble_mesh_time_scene_server_cb_param_t::ctx

esp_ble_mesh_time_scene_client_set_state_t::scene_delete
esp_ble_mesh_time_scene_client_set_state_t::scene_change
esp_ble_mesh_time_scene_client_set_state_t::scene_register

ESP_BLE_MESH_TAMPER_ERROR (C macro), 631
ESP_BLE_MESH_TAMPER_WARNING (C macro), 631
ESP_BLE_MESH_TIME_CLINET (C macro), 719
ESP_BLE_MESH_TIME_NONE (C macro), 719
ESP_BLE_MESH_TIME_RELAY (C macro), 719

esp_ble_mesh_time_role_set_t (C++ struct), 698
esp_ble_mesh_time_role_set_t::time_role
esp_ble_mesh_time_role_set_t::ESP_BLE_MESH_TIME_CLINET
esp_ble_mesh_time_role_set_t::ESP_BLE_MESH_TIME_NONE
esp_ble_mesh_time_role_set_t::ESP_BLE_MESH_TIME_CLINET

esp_ble_mesh_time_scene_server_cb_event_t::ESP_BLE_MESH_TIME_SCENE_SERVER_STATE_CHANGE_EVT
esp_ble_mesh_time_scene_server_cb_event_t::ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_SET_MSG_EVT
esp_ble_mesh_time_scene_server_cb_event_t::ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_GET_MSG_EVT
esp_ble_mesh_time_scene_server_cb_event_t::ESP_BLE_MESH_TIME_SCENE_SERVER_EVT_MAX

esp_ble_mesh_time_scene_client_status_cb_t::time_status
esp_ble_mesh_time_scene_client_status_cb_t::time_role_status
esp_ble_mesh_time_scene_client_status_cb_t::tai_utc_delta_status
esp_ble_mesh_time_scene_client_status_cb_t::scheduler_status
esp_ble_mesh_time_scene_client_status_cb_t::scheduler_act_status
esp_ble_mesh_time_scene_client_status_cb_t::scene_status
esp_ble_mesh_time_scene_client_status_cb_t::scene_register_status

esp_ble_mesh_time_scene_client_status_cb_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:

esp_ble_mesh_time_scene_server_cb_event_t:
esp_ble_mesh_time_set_t::tai_utc_delta (C++ member), 697
esp_ble_mesh_time_set_t::time_authority (C++ member), 697
esp_ble_mesh_time_set_t::time_zone_offset (C++ member), 697
esp_ble_mesh_time_set_t::time_zone_offset_new (C++ member), 697
esp_ble_mesh_time_set_t::tai_seconds (C++ member), 697
esp_ble_mesh_time_set_t::sub_second (C++ member), 697
esp_ble_mesh_time_scene_server_state_change_t::time_zone_set (C++ struct), 704
esp_ble_mesh_time_scene_server_state_change_t::time_status (C++ member), 704
esp_ble_mesh_time_scene_server_state_change_t::time_set (C++ member), 704
esp_ble_mesh_time_scene_server_state_change_t::time_role_set (C++ member), 704
esp_ble_mesh_time_scene_server_state_change_t::tai_utc_delta_set (C++ member), 704
esp_ble_mesh_time_scene_server_state_change_t::scheduler_act_set (C++ member), 704
esp_ble_mesh_time_scene_server_state_change_t::scene_set (C++ member), 704
esp_ble_mesh_time_scene_server_state_change_t::scene_recall (C++ member), 704
esp_ble_mesh_time_scene_server_state_change_t::scene_delete (C++ member), 704
esp_ble_mesh_time_scene_server_recv_status_msg_t::status (C++ member), 704
esp_ble_mesh_time_scene_server_recv_status_msg_t::state_change (C++ member), 704
esp_ble_mesh_time_scene_server_recv_status_msg_t::set (C++ member), 704
esp_ble_mesh_time_scene_server_recv_status_msg_t::get (C++ member), 704
esp_ble_mesh_time_scene_server_recv_set_msg_t::tai_seconds (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::sub_second (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::time_authority (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::time_zone (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::time_role (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::time (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::tai_utc_delta (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::scheduler_act (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::scene_recall (C++ member), 697
esp_ble_mesh_time_scene_server_recv_set_msg_t::scene_delete (C++ member), 697
esp_ble_mesh_time_scene_server_recv_get_msg_t::scheduler_act (C++ member), 697
esp_ble_mesh_time_scene_server_recv_get_msg_t::scene_recall (C++ member), 697
esp_ble_mesh_time_scene_server_recv_get_msg_t::scene_delete (C++ member), 697

esp_ble_mesh_time_status_cb_t::status (C++ member), 697
esp_ble_mesh_time_status_cb_t::state_change (C++ member), 697
esp_ble_mesh_time_status_cb_t::set (C++ member), 697
esp_ble_mesh_time_status_cb_t::get (C++ member), 697

esp_ble_mesh_time_state_t::uncertainty (C++ member), 704
esp_ble_mesh_time_state_t::time_zone_offset_new (C++ member), 704
esp_ble_mesh_time_state_t::time_zone_offset_curr (C++ member), 704
esp_ble_mesh_time_state_t::time_role (C++ member), 704
esp_ble_mesh_time_state_t::time_authority (C++ member), 704
esp_ble_mesh_time_state_t::time (C++ member), 704
esp_ble_mesh_time_state_t::tai_zone_change (C++ member), 704
esp_ble_mesh_time_state_t::time_zone_change (C++ member), 704
esp_ble_mesh_time_state_t::tai_utc_delta (C++ member), 704

esp_ble_mesh_time_setup_srv_t::state (C++ member), 704
esp_ble_mesh_time_setup_srv_t::rsp_ctrl (C++ member), 704
esp_ble_mesh_time_setup_srv_t::model (C++ member), 704
esp_ble_mesh_time_setup_srv_t::state_change (C++ member), 704
esp_ble_mesh_time_setup_srv_t::time_role (C++ member), 704
esp_ble_mesh_time_setup_srv_t::tai_seconds (C++ member), 704
esp_ble_mesh_time_setup_srv_t::sub_second (C++ member), 704
esp_ble_mesh_time_setup_srv_t::time_authority (C++ member), 704
esp_ble_mesh_time_setup_srv_t::tai_utc_delta (C++ member), 704

esp_ble_mesh_time_setup_srv_t::model (C++ member), 704
esp_ble_mesh_time_setup_srv_t::state (C++ member), 704
esp_ble_mesh_time_setup_srv_t::time_role (C++ member), 704
esp_ble_mesh_time_setup_srv_t::tai_seconds (C++ member), 704
esp_ble_mesh_time_setup_srv_t::sub_second (C++ member), 704
esp_ble_mesh_time_setup_srv_t::time_authority (C++ member), 704
esp_ble_mesh_time_setup_srv_t::tai_utc_delta (C++ member), 704
esp_ble_secure_list_flush
(C++ function), 241
esp_ble_secure_type_t (C++ enum), 298
esp_ble_secure_type_t::BLE_SECURE_TYPE_UNSECURED
(C++ enumerator), 298
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT
(C++ enumerator), 298
esp_ble_secure_type_t::BLE_SECURE_TYPE_A2DP
(C++ enumerator), 298
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP
(C++ enumerator), 298
esp_ble_secure_type_t::BLE_SECURE_TYPE_HSP
(C++ enumerator), 298
esp_ble_secure_type_t::BLE_SECURE_TYPE_HSP_HFP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_HSP_EAT
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_EAT
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_HSP_A2DP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_A2DP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_HSP_HFP_A2DP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT_A2DP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT_HFP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT_HSP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT_HSP_HFP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT_HSP_A2DP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT_HFP_A2DP
(C++ enumerator), 299
esp_ble_secure_type_t::BLE_SECURE_TYPE_EAT_HSP_HFP_A2DP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_BLUEZ
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_EAT_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_A2DP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_A2DP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_A2DP_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_A2DP_HSP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_A2DP_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_A2DP_HFP_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_A2DP_HFP_HSP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_A2DP_HFP_HSP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_A2DP_HFP_HSP_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_A2DP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_HFP_EAT_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_HFP_EAT_HFP_EAT
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_HFP_EAT_HFP_EAT_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_HFP_EAT_HFP_EAT_HFP_HFP
(C++ enumerator), 300
esp_ble_secure_type_t::BLE_SECURE_TYPE_HFP_HSP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_EAT_HFP_EAT_HSP_HFP_HFP_HFP_EAT_HFP_EAT_HFP_HFP_HFP
(C++ enumerator), 300

Espressif Systems

Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151

2621
esp_blufi_callbacks_t::decrypt_func
(C++ member), 361

esp_blufi_callbacks_t::encrypt_func
(C++ member), 361

esp_blufi_callbacks_t::event_cb (C++ member), 361

esp_blufi_callbacks_t::negotiate_data_handler
(C++ member), 361

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_REQ_DISCONNECT_FROM_AP
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_REQ_CONNECT_TO_AP
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_REPORT_ERROR
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_USERNAME
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_STA_SSID
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_STA_PASSWD
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_STA_BSSID
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_SOFTAP_SSID
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_SERVER_CERT
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_CUSTOM_DATA
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_CLIENT_CERT
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_RECV_CA_CERT
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_INIT_FINISH
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_DEINIT_FINISH
(C++ enumerator), 363

esp_blufi_cb_event_t::ESP_BLUFI_EVENT_BLE_CONNECT
(C++ enumerator), 363

esp_blufi_callbacks_t::negotiate_data_handler
(C++ member), 361

esp_blufi_ap_record_t::struct, 361

esp_blufi_ap_record_t::rssi (C++ member), 361

ESP_BLUFI_BDADDR_LEN (C macro), 362

esp_blufi_cb_event_t (C++ type), 362

esp_blufi_callbacks_t::struct, 361

esp_blufi_callbacks_t::checksum_func
(C++ member), 362

esp_blufi_callbacks_t::decrypt_func
(C++ member), 362

esp_blufi_callbacks_t::encrypt_func
(C++ member), 362

esp_blufi_callbacks_t::event_cb (C++ member), 361

esp_bluficallbacks_t::negotiate_data_handler
(C++ member), 361

esp_blufi_callbacks_t::decrypt_func
(C++ member), 361
<table>
<thead>
<tr>
<th>C++ member</th>
<th>C++ struct</th>
<th>C++ union</th>
<th>C++ enum</th>
<th>C++ typedef</th>
</tr>
</thead>
<tbody>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ union)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ struct)</td>
<td>359</td>
</tr>
<tr>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param</td>
<td>esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::pkey_len</td>
<td>esp_blufi_cb_event_t::ESP_BLUFI_EVENT_SET_WIFI_OPMODE</td>
<td>(C++ member)</td>
<td>359</td>
</tr>
</tbody>
</table>
esp_blufi_cb_param_t::client_cert (C++ member), 355
esp_blufi_cb_param_t::client_pkey (C++ member), 355
esp_blufi_cb_param_t::connect (C++ member), 354
esp_blufi_cb_param_t::custom_data (C++ member), 355
esp_blufi_cb_param_t::deinit_finish (C++ member), 354
esp_blufi_cb_param_t::disconnect (C++ member), 354
esp_blufi_cb_param_t::init_finish (C++ member), 354
esp_blufi_cb_param_t::report_error (C++ member), 355
esp_blufi_cb_param_t::server_cert (C++ member), 355
esp_blufi_cb_param_t::server_pkey (C++ member), 355
esp_blufi_cb_param_t::softap_auth_mode (C++ member), 354
esp_blufi_cb_param_t::softap_channel (C++ member), 354
esp_blufi_cb_param_t::softap_max_conn_num (C++ member), 354
esp_blufi_cb_param_t::softap_passwd (C++ member), 354
esp_blufi_cb_param_t::softap_ssid (C++ member), 354
esp_blufi_extra_info_t::sta_bssid (C++ member), 354
esp_blufi_extra_info_t::sta_bssid_set (C++ member), 354
esp_blufi_extra_info_t::sta_connect_rssi (C++ member), 354
esp_blufi_extra_info_t::sta_connect_reason (C++ member), 354
esp_blufi_extra_info_t::sta_connect_reason_set (C++ member), 354
esp_blufi_extra_info_t::sta_connect_rssi_set (C++ member), 354
esp_blufi_extra_info_t::sta_conn_end_reason (C++ member), 354
esp_blufi_extra_info_t::sta_conn_end_reason_set (C++ member), 354
esp_blufi_extra_info_t::sta_passwd (C++ member), 354
esp_blufi_extra_info_t::sta_password (C++ member), 354
esp_blufi_extra_info_t::sta_password_len (C++ member), 354
esp_blufi_extra_info_t::sta_max_conn_retry (C++ member), 354
esp_blufi_extra_info_t::sta_max_conn_retry_set (C++ member), 354

(C++ type), 362

(C++ struct), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360

(C++ member), 360
索引

esp_bt_connection_mode_t (C++ enum), 380
486
esp_bt_connection_mode_t::ESP_BT_CONNECTABLE
esp_bt_controller_get_status (C++ func(C++ enumerator), 380
tion), 487
esp_bt_connection_mode_t::ESP_BT_NON_CONNECTABLE
esp_bt_controller_init (C++ function), 486
(C++ enumerator), 380
esp_bt_controller_mem_release (C++ funcESP_BT_CONTROLLER_CONFIG_MAGIC_VAL (C
tion), 487
macro), 491
esp_bt_controller_status_t (C++ enum),
esp_bt_controller_config_t (C++ struct),
492
489
esp_bt_controller_status_t::ESP_BT_CONTROLLER_STA
esp_bt_controller_config_t::auto_latency
(C++ enumerator), 492
esp_bt_controller_status_t::ESP_BT_CONTROLLER_STA
(C++ member), 490
esp_bt_controller_config_t::ble_max_conn
(C++ enumerator), 492
(C++ member), 490
esp_bt_controller_status_t::ESP_BT_CONTROLLER_STA
esp_bt_controller_config_t::ble_sca
(C++ enumerator), 492
esp_bt_controller_status_t::ESP_BT_CONTROLLER_STA
(C++ member), 490
esp_bt_controller_config_t::bt_legacy_auth_vs_evt
(C++ enumerator), 492
(C++ member), 490
esp_bt_dev_get_address (C++ function), 242
esp_bt_controller_config_t::bt_max_acl_conn
esp_bt_dev_set_device_name (C++ function),
(C++ member), 490
242
esp_bt_controller_config_t::bt_max_sync_conn
esp_bt_dev_type_t (C++ enum), 240
(C++ member), 490
esp_bt_dev_type_t::ESP_BT_DEVICE_TYPE_BLE
esp_bt_controller_config_t::bt_sco_datapath (C++ enumerator), 240
(C++ member), 490
esp_bt_dev_type_t::ESP_BT_DEVICE_TYPE_BREDR
esp_bt_controller_config_t::controller_debug_flag
(C++ enumerator), 240
(C++ member), 490
esp_bt_dev_type_t::ESP_BT_DEVICE_TYPE_DUMO
(C++ enumerator), 240
esp_bt_controller_config_t::controller_task_prio
esp_bt_discovery_mode_t (C++ enum), 380
(C++ member), 489
esp_bt_controller_config_t::controller_task_stack_size
esp_bt_discovery_mode_t::ESP_BT_GENERAL_DISCOVERA
(C++ member), 489
(C++ enumerator), 380
esp_bt_controller_config_t::hci_uart_baudrate
esp_bt_discovery_mode_t::ESP_BT_LIMITED_DISCOVERA
(C++ member), 489
(C++ enumerator), 380
esp_bt_controller_config_t::hci_uart_noesp_bt_discovery_mode_t::ESP_BT_NON_DISCOVERABLE
(C++ member), 489
(C++ enumerator), 380
esp_bt_controller_config_t::hli (C++ esp_bt_duplicate_exceptional_subcode_type_t
member), 490
(C++ enum), 303
esp_bt_controller_config_t::magic
esp_bt_duplicate_exceptional_subcode_type_t::ESP_
(C++ member), 490
(C++ enumerator), 303
esp_bt_controller_config_t::mesh_adv_size
esp_bt_duplicate_exceptional_subcode_type_t::ESP_
(C++ member), 489
(C++ enumerator), 303
esp_bt_controller_config_t::mode (C++ esp_bt_duplicate_exceptional_subcode_type_t::ESP_
member), 490
(C++ enumerator), 303
esp_bt_controller_config_t::normal_adv_size
esp_bt_eir_data_t (C++ struct), 376
(C++ member), 489
esp_bt_eir_data_t::fec_required (C++
esp_bt_controller_config_t::pcm_polar
member), 376
(C++ member), 490
esp_bt_eir_data_t::flag (C++ member), 376
esp_bt_controller_config_t::pcm_role
esp_bt_eir_data_t::include_txpower
(C++ member), 490
(C++ member), 376
esp_bt_controller_config_t::scan_duplicate_mode
esp_bt_eir_data_t::include_uuid (C++
(C++ member), 489
member), 376
esp_bt_controller_config_t::scan_duplicate_type
esp_bt_eir_data_t::manufacturer_len
(C++ member), 489
(C++ member), 376
esp_bt_controller_config_t::send_adv_reserved_size
esp_bt_eir_data_t::p_manufacturer_data
(C++ member), 489
(C++ member), 376
esp_bt_controller_deinit (C++ function), esp_bt_eir_data_t::p_url (C++ member),
486
376
esp_bt_controller_disable (C++ function), esp_bt_eir_data_t::url_len (C++ member),
487
376
esp_bt_controller_enable (C++ function), ESP_BT_EIR_FLAG_BREDR_NOT_SPT (C macro),

Espressif Systems

2626
Submit Document Feedback

Release v5.1-dev-2066-g7869f4e151


ESP_BT_HF_CLIENT_OPERATOR_NAME_LEN (C macro), 444
ESP_BT_HF_CLIENT_NUMBER_LEN (C macro), 444
ESP_BT_HF_AT_SEND_XAPL_LEN (C macro), 444
ESP_BT_HF_CLIENT_OPERATOR_NAME_LEN (C macro), 444
ESP_BT_HF_CLIENT_NUMBER_LEN (C macro), 444
esp_cpu_watchpoint_trigger_t::ESP_CPU_WATCHPOINT_TRIGGER
(C++ function), 1143
esp_crt_bundle_attach (C++ function), 191
esp_crt_bundle_detach (C++ function), 191
esp_crt_bundle_set (C++ function), 191
esp_deep_sleep (C++ function), 2046
esp_deep_sleep_disable_rom_logging
(C++ function), 2047
esp_deep_sleep_start (C++ function), 2046
esp_deep_sleep_wake_stub_fn_t (C++ function), 2047
ESP_DEFAULT_GATT_IF (C macro), 237
esp_default_wake_deep_sleep (C++ function), 2047
deregister_freertos_idle_hook
(C++ function), 1946
deregister_freertos_idle_hook_for_cpu
(C++ function), 1946
deregister_freertos_tick_hook
(C++ function), 1946
duplicate_info_t (C++ type), 292
duplicate_scan_exceptional_list_type_t
(C++ enum), 304
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROXY_SRV_ADV_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_BEACON_TYPE_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ALL_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROXY
(C++ function), 1946
ESP_EARLY_LOGI (C macro), 1995
ESP_EARLY_LOGE (C macro), 1995
ESP_EARLY_LOGD (C macro), 1995
ESP_EARLY_LOGV (C macro), 1995
esp_efuse_block_t::EFUSE_BLK1 (C++ enumeration), 1783
esp_efuse_block_t::EFUSE_BLK2 (C++ enumeration), 1783
esp_efuse_block_t::EFUSE_BLK3 (C++ enumeration), 1783
esp_efuse_block_t::EFUSE_BLK_ENCRYPT_FLASH
(C++ enumeration), 1783
esp_efuse_block_t::EFUSE_BLK_KEY0
(C++ enumeration), 1783
esp_efuse_block_t::EFUSE_BLK_KEY1
(C++ enumeration), 1783
esp_efuse_block_t::EFUSE_BLK_KEY2
(C++ enumeration), 1783
deregister_freertos_idle_hook
for_cpu (C++ function), 1946
derive_local_mac (C++ function), 2004
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ALL_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_BEACON_TYPE_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROXY
(C++ function), 1946
ESP_EARLY_LOGD (C macro), 1995
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_SRV_ADV_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_BEACON_TYPE_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_LINK_ID_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_MESHPROXY_ADDR_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_MESHPROXY_ALL_LIST
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_MESHPROXY_MESHPROXY
(C++ function), 1946
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_MESHPROXY_MESHPROXY_ADDR_LIST
(C++ function), 1946
esp_efuse_block_t::EFUSE_BLK0
(C++ enumeration), 1782
esp_efuse_batch_write_begin (C++ function), 1789
esp_efuse_batch_write_cancel (C++ function), 1790
esp_efuse_batch_write_commit (C++ function), 1790
esp_efuse_block_is_empty (C++ function), 1790
duplicate_scan_exceptional_list_type_t::ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESHPROXY_MESHPROXY_MESHPROXY_MESHPROXY
(C++ function), 1946
esp_efuse_block_t (C++ enum), 1782
esp_efuse_block_t::EFUSE_BLK0 (C++ enumeration), 1782
esp_show_efuse1 (C function), 1946
ESP_ERR_NVS_ENCR_NOT_SUPPORTED (C macro), 1670
ESP_ERR_NVS_INVALID_HANDLE (C macro), 1669
ESP_ERR_NVS_INVALID_LENGTH (C macro), 1669
ESP_ERR_NVS_INVALID_NAME (C macro), 1669
ESP_ERR_NVS_INVALID_STATE (C macro), 1669
ESP_ERR_NVS_KEY_TOO_LONG (C macro), 1669
ESP_ERR_NVS_KEYS_NOT_INITIALIZED (C macro), 1670
ESP_ERR_NVS_NEW_VERSION_FOUND (C macro), 1669
ESP_ERR_NVS_NO_FREE_PAGES (C macro), 1669
ESP_ERR_NVS_NOT_ENOUGH_SPACE (C macro), 1669
ESP_ERR_NVS_NOT_FOUND (C macro), 1668
ESP_ERR_NVS_NOT_INITIALIZED (C macro), 1668
ESP_ERR_NVS_PAGE_FULL (C macro), 1669
ESP_ERR_NVS_PART_NOT_FOUND (C macro), 1669
ESP_ERR_NVS_READ_ONLY (C macro), 1669
ESP_ERR_NVS_REMOVE_FAILED (C macro), 1669
ESP_ERR_NVS_TYPE_MISMATCH (C macro), 1669
ESP_ERR_NVS_VALUE_TOO_LONG (C macro), 1669
ESP_ERR_NVS_WRONG_ENCRYPTION (C macro), 1670
ESP_ERR_NVS_XTS_CFG_FAILED (C macro), 1670
ESP_ERR_NVS_XTS_CFG_NOT_FOUND (C macro), 1670
ESP_ERR_NVS_XTS_DECR_FAILED (C macro), 1669
ESP_ERR_NVS_XTS_ENCR_FAILED (C macro), 1669
ESP_ERR_NVS_BASE (C macro), 2022
ESP_ERR_NVS_PARTITION_CONFICT (C macro), 2022
ESP_ERR_NVS.Rollback_FAILED (C macro), 2022
ESP_ERR_NVS.Rollback_INVALID_STATE (C macro), 2022
ESP_ERR_NVS_SELECT_INFO_INVALID (C macro), 2022
ESP_ERR_NVS.Small_SEC_VER (C macro), 2022
ESP_ERR_NVS_VERIFY_FAILED (C macro), 2022
esp_err_t (C++ type), 1796
ESP_ERR_TIMEOUT (C macro), 1795
esp.err_to_name (C++ function), 1794
esp.err_to_name_r (C++ function), 1794
ESP_ERR_ULP_BASE (C macro), 2096
ESP_ERR_ULP_BRANCH_OUT_OF_RANGE (C macro), 2097
ESP_ERR_ULP_DUPLICATE_LABEL (C macro), 2097
ESP_ERR_ULP_INVALID_LOAD_ADDR (C macro), 2097
ESP_ERR_ULP_SIZE_TOO_BIG (C macro), 2096
ESP_ERR_ULP_UNDEFINED_LABEL (C macro), 2097
ESP_ERR_WIFI_BASE (C macro), 1796
ESP_ERR_WIFI_CONN (C macro), 851
ESP_ERR_WIFI_IF (C macro), 851
ESP_ERR_WIFI_INIT_STATE (C macro), 852
ESP_ERR_WIFI_MAC (C macro), 852
ESP_ERR_WIFI_MODE (C macro), 851
ESP_ERR_WIFI.NOT_ASSOC (C macro), 852
ESP_ERR_WIFI.NOT_CONNECT (C macro), 852
ESP_ERR_WIFI.NOT_INIT (C macro), 851
ESP_ERR_WIFI.NOT_STARTED (C macro), 851
ESP_ERR_WIFI.NOT_STOPPED (C macro), 851
ESP_ERR_WIFI.NVS (C macro), 851
ESP_ERR_WIFI_PASSWD (C macro), 852
ESP_ERR_WIFI_POST (C macro), 852
ESP_ERR_WIFI_SSID (C macro), 852
ESP_ERR_WIFI_STATE (C macro), 851
ESP_ERR_WIFI_STOP_STATE (C macro), 852
ESP_ERR_WIFI_TIMEOUT (C macro), 852
ESP_ERR_WIFI_TX_DISALLOW (C macro), 852
ESP_ERR_WIFI.WAKE_FAIL (C macro), 852
ESP_ERR_WIFI.WOULD_BLOCK (C macro), 852
ESP_ERR_WOLFSSL_CERT_VERIFY_FAILED (C macro), 155
ESP_ERR_WOLFSSL_CTX_SETUP_FAILED (C macro), 155
ESP_ERR_WOLFSSL_KEY_VERIFY_FAILED (C macro), 155
ESP_ERR_WOLFSSL_SSL_CONF_ALPN_PROTOCOLS_FAILED (C macro), 155
ESP_ERR_WOLFSSL_SSL_HANDSHAKE_FAILED (C macro), 155
ESP_ERR_WOLFSSL_SSL_SET_HOSTNAME_FAILED (C macro), 155
ESP_ERR_WOLFSSL_SSL_SETUP_FAILED (C macro), 156
ESP_ERR_WOLFSSL_SSL_WRITE_FAILED (C macro), 156
ESP_ERROR_CHECK (C macro), 1796
ESP_ERROR_CHECK_WITHOUT_ABORT (C macro), 1796
esp.esptouch.set.timeout (C++ function), 831
esp.eth_config.t (C++ struct), 900
esp.eth_config.t::check_link_period_ms (C++ member), 901
esp.eth_config.t::mac (C++ member), 901
esp.eth_config.t::on_lowlevel_deinit_done (C++ member), 901
esp.eth_config.t::on_lowlevel_init_done (C++ member), 901
esp.eth_config.t::phy (C++ member), 901
esp.eth_config.t::read_phy_reg (C++ member), 901
esp.eth_config.t::stack_input (C++ member), 901
esp.eth_config.t::write_phy_reg (C++ member), 902
esp.eth_decrease_reference (C++ function),
esp_eth_del_netif_glue (C++ function), 919
esp_eth_driver_install (C++ function), 897
esp_eth_driver_uninstall (C++ function), 897
esp_eth_handle_t (C++ type), 902
esp_eth_increase_reference (C++ function), 900
esp_eth_io_cmd_t (C++ enum), 902
esp_eth_io_cmd_t::ETH_CMD_CUSTOM_MAC_CMDS (C++ enumerator), 903
esp_eth_io_cmd_t::ETH_CMD_CUSTOM_PHY_CMDS (C++ enumerator), 903
esp_eth_io_cmd_t::ETH_CMD_G_AUTONEGO (C++ enumerator), 902
esp_eth_io_cmd_t::ETH_CMD_G_DUPELEX_MODE (C++ enumerator), 903
esp_eth_io_cmd_t::ETH_CMD_G_MAC_ADDR (C++ enumerator), 902
esp_eth_io_cmd_t::ETH_CMD_G_PHY_ADDR (C++ enumerator), 902
esp_eth_io_cmd_t::ETH_CMD_G_SPEED (C++ enumerator), 902
esp_eth_mac_t::set_link (C++ member), 909
esp_eth_mac_t::set_mediator (C++ member), 906
esp_eth_mac_t::set_peer_pause_ability (C++ member), 909
esp_eth_mac_t::set_promiscuous (C++ member), 909
esp_eth_mac_t::set_speed (C++ member), 908
esp_eth_mac_t::set_link (C++ member), 906
esp_eth_mac_t::set_promiscuous (C++ member), 906
esp_eth_mac_t::set_speed (C++ member), 907
esp_eth_mac_t::set_promiscuous (C++ member), 907
esp_eth_mac_t::set_state (C++ member), 907
esp_eth_mac_t::set_speed (C++ member), 907
esp_eth_mac_t::stop (C++ member), 906
esp_eth_mac_t::transmit (C++ member), 907
esp_eth_mac_t::transmit_vargs (C++ member), 907
esp_eth_mac_t::write_phy_reg (C++ member), 908
esp_eth_mac_t (C++ type), 911
esp_eth_mediator_s (C++ struct), 903
esp_eth_mediator_s::on_state_changed (C++ member), 904
esp_eth_mediator_s::phy_reg_read (C++ member), 904
esp_eth_mediator_s::phy_reg_write (C++ member), 904
esp_eth_mediator_s::stack_input (C++ member), 904
esp_eth_mediator_t (C++ type), 904
esp_netif_glue_handle_t (C++ type), 919
esp_netif_glue_handle_t (C++ type), 919
esp_phy_802_3_basic_phy_deinit (C++ function), 917
esp_phy_802_3_basic_phy_deinit (C++ function), 917
esp_phy_802_3_basic_phy_init (C++ function), 917
esp_phy_802_3_basic_phy_init (C++ function), 917
esp_phy_802_3_detect_phy_addr (C++ function), 917
esp_phy_802_3_obj_config_init (C++ function), 918
esp_phy_802_3_obj_config_init (C++ function), 918
esp_phy_802_3_read_manufac_info (C++ function), 918
esp_phy_802_3_read_manufac_info (C++ function), 918
esp_phy_802_3_read oui (C++ function), 918
esp_phy_802_3_read oui (C++ function), 918
esp_phy_802_3_reset_hw (C++ function), 917
ESP_ETH_PHY_ADDR_AUTO (C macro), 916
esp_phy_into_phy_802_3 (C++ function), 916
esp_phy_new_dp83848 (C++ function), 912
esp_phy_new_dp83848 (C++ function), 912
esp_phy_new_ip101 (C++ function), 912
esp_phy_new_ip101 (C++ function), 912
esp_phy_new_ksz80xx (C++ function), 913
esp_phy_new_ksz80xx (C++ function), 913
esp_phy_new_lan87xx (C++ function), 912
esp_phy_new_lan87xx (C++ function), 912
esp_phy_new_til8201 (C++ function), 912
esp_phy_new_til8201 (C++ function), 912
esp_phy_s (C++ struct), 913
esp_phy_s::advertise_pause_ability (C++ member), 915
esp_phy_s::advertise_pause_ability (C++ member), 915
esp_phy_s::autonego_ctrl (C++ member), 914
esp_eth_phy_s::del (C++ member), 916
esp_eth_phy_s::get_addr (C++ member), 914
esp_eth_phy_s::get_link (C++ member), 914
esp_eth_phy_s::init (C++ member), 913
esp_eth_phy_s::loopback (C++ member), 915
esp_eth_phy_s::pwrct1 (C++ member), 914
esp_eth_phy_s::reset (C++ member), 913
esp_eth_phy_s::reset_hw (C++ member), 913
esp_eth_phy_s::set_addr (C++ member), 914
esp_eth_phy_s::set_duplex (C++ member), 915
esp_eth_phy_s::setmediator (C++ member), 913
esp_eth_phy_s::set_speed (C++ member), 915
esp_eth_phy_t (C++ type), 916
esp_eth_start (C++ function), 898
esp_event_state_t (C++ enum), 904
esp_event_state_t::ETH_STATE_DEINIT (C++ enumerator), 904
esp_event_state_t::ETH_STATEDuplex (C++ enumerator), 905
esp_event_state_t::ETH_STATE_LINK (C++ enumerator), 904
esp_event_state_t::ETH_STATE_LLINIT (C++ enumerator), 904
esp_event_state_t::ETH_STATE_PAUSE (C++ enumerator), 905
esp_event_state_t::ETH_STATESPEED (C++ enumerator), 904
esp_eth_stop (C++ function), 898
esp_eth_transmit (C++ function), 899
esp_eth_transmit_vargs (C++ function), 899
esp_eth_update_input_path (C++ function), 898
ESP_EVENT_ANY_BASE (C macro), 1814
ESP_EVENT_ANY_ID (C macro), 1814
ESP_EVENTDECLARE_BASE (C macro), 1814
ESP_EVENTDEFINE_BASE (C macro), 1814
esp_event_dump (C function), 1812
esp_event_handler_instance_register (C function), 1809
esp_event_handler_instance_register with (C function), 1808
esp_event_handler_instance_t (C++ type), 1814
esp_event_handler_instance_unregister (C++ function), 1810
esp_event_handler_instance_unregister with flash_encrypt_is_write_protected (C++ function), 1810
esp_event_handler_register (C++ function), 1807
esp_event_handler_register with (C function), 1807
esp_event_handler_t (C++ type), 1814
esp_event_handler_unregister (C++ function), 1809
esp_event_handler_unregister with (C++ function), 1810
esp_event_isr_post (C++ function), 1811
esp_event_isr_post_to (C++ function), 1812
esp_event_loop_args_t (C++ struct), 1813
esp_event_loop_args_t::queue_size (C++ member), 1813
esp_event_loop_args_t::task_core_id (C++ member), 1814
esp_event_loop_args_t::task_name (C++ member), 1813
esp_event_loop_args_t::task_priority (C++ member), 1813
esp_event_loop_args_t::task_stack_size (C++ member), 1813
esp_event_loop_create (C++ function), 1806
esp_event_loop_create_default (C++ function), 1806
esp_event_loop_delete (C++ function), 1806
esp_event_loop_delete_default (C++ function), 1806
esp_event_loop_handle_t (C++ type), 1814
esp_event_loop_run (C++ function), 1806
esp_event_post (C++ function), 1811
esp_event_post_to (C++ function), 1811
ESP_EXECUTE_EXPRESSION_WITH_STACK (C macro), 1761
esp_execute_shared_stack_function (C++ function), 1761
ESP_FAIL (C macro), 1795
esp_fill_random (C++ function), 2037
flash_chip_driver_initialized (C++ function), 1700
flash_enc_mode_t (C++ enum), 1726
flash_enc_mode_t::ESP_FLASH_ENC_MODEDEVELOPMENT (C++ enumerator), 1726
flash_enc_mode_t::ESP_FLASH_ENC_MODEDISABLED (C++ enumerator), 1726
flash_enc_mode_t::ESP_FLASH_ENC_MODERELEASE (C++ enumerator), 1726
flash_encrypt_check_and_update (C++ function), 1724
flash_encrypt_contents (C++ function), 1724
flash_encrypt_enable (C++ function), 1724
flash_encrypt_decrypt (C++ function), 1724
flash_encrypt_is_write_protected (C++ function), 1725
flash_encrypt_region (C++ function), 1725
flash_encrypt_state (C++ function), 1724
flash_encryption_enable_secure_features (C++ function), 1725
flash_encryption_enabled (C++ function), 1724
esp_gap_search_evt_t::ESP_GAP_SEARCH_INQ_RES_EVT
esp_gap_search_evt_t::ESP_GAP_SEARCH_INQ_DISCARD_NUM_EVT
esp_gap_search_evt_t::ESP_GAP_SEARCH_INQ_CMPL_EVT
esp_gap_search_evt_t::ESP_GAP_SEARCH_DISC_RES_EVT
esp_gap_search_evt_t::ESP_GAP_SEARCH_DISC_CMPL_EVT
esp_gap_search_evt_t::ESP_GAP_SEARCH_DISC_BLE_RES_EVT
esp_gap_search_evt_t::ESP_GAP_SEARCH_DISC_CMPL_EVT

esp_gap_search_evt_t

esp_gap_conn_params_t::timeout
esp_gap_conn_params_t::latency

esp_gap_ble_set_channels
esp_gap_ble_set_authorization
ESP_GAP_BLE_CHANNELS_LEN
esp_gap_ble_channels

esp_gap_ble_cb_t

esp_gap_ble_cb_event_t::ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SET_PREFERRED_DEFAULT_PHY_COMPLETE_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SET_EXT_SCAN_PARAMS_COMPLETE_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SET_CHANNELS_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SEC_REQ_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SCAN_TIMEOUT_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT
esp_gap_ble_cb_event_t::ESP_GAP_BLE_SCAN_START_COMPLETE_EVT

esp_gatt_conn_reason_t::ESP_GATT_CONN_TIMEOUT
esp_gatt_conn_reason_t::ESP_GATT_CONN_TERMINATE_PEER_USER
esp_gatt_conn_reason_t::ESP_GATT_CONN_TERMINATE_LOCAL_HOST
esp_gatt_conn_reason_t::ESP_GATT_CONN_NONE
esp_gatt_conn_reason_t::ESP_GATT_CONN_LMP_TIMEOUT
esp_gatt_conn_reason_t::ESP_GATT_CONN_L2C_FAILURE
esp_gatt_conn_reason_t::ESP_GATT_CONN_CONN_CANCEL

esp_gatt_conn_reason_t

esp_gatt_conn_params_t::timeout
esp_gatt_conn_params_t::latency

esp_gatt_char_prop_t

esp_gatt_auth_req_t::ESP_GATT_AUTH_REQ_SIGNED_NO_MITM
esp_gatt_auth_req_t::ESP_GATT_AUTH_REQ_SIGNED_MITM
esp_gatt_auth_req_t::ESP_GATT_AUTH_REQ_NONE
esp_gatt_auth_req_t::ESP_GATT_AUTH_REQ_NO_MITM
esp_gatt_auth_req_t::ESP_GATT_AUTH_REQ_MITM

esp_gatt_auth_req_t

ESP_GATT_ATTR_HANDLE_MAX

ESP_GATT_CHAR_PROP_BIT_WRITE
ESP_GATT_CHAR_PROP_BIT_READ
ESP_GATT_CHAR_PROP_BIT_NOTIFY
ESP_GATT_CHAR_PROP_BIT_INDICATE
ESP_GATT_CHAR_PROP_BIT_EXT_PROP
ESP_GATT_CHAR_PROP_BIT_BROADCAST
ESP_GATT_CHAR_PROP_BIT_AUTH

ESP_GATT_BODY_SENSOR_LOCATION
ESP_GATT_AUTO_RSP
esp_gatt_conn_reason_t::ESP_GATT_CONN_UNKNOWN (C++ enumerator), 315
esp_gatt_srvc_id_t::id (C++ member), 305
esp_gatt_srvc_id_t::is_primary (C++ member), 305
esp_gatt_status_t::ESP_GATT_ALREADY_OPEN (C++ enumerator), 317
esp_gatt_status_t::ESP_GATT_APP_RSP (C++ enumerator), 317
esp_gatt_status_t::ESP_GATT_AUTH_FAIL (C++ enumerator), 317
esp_gatt_status_t::ESP_GATT_BUSY (C++ enumerator), 316
esp_gatt_status_t::ESP_GATT_CANCEL (C++ enumerator), 319
ESP_GATT_HEART_RATE_CNTL_POINT macro, 313
ESP_GATT_HEART_RATE_MEAS (C macro), 313
esp_gatt_id_t (C++ struct), 304
esp_gatt_id_t::inst_id (C++ member), 305
esp_gatt_id_t::uuid (C++ member), 305
ESP_GATT_IF_NONE (C macro), 315
esp_gatt_if_t (C++ type), 315
ESP_GATT_ILLEGAL_HANDLE (C macro), 314
ESP_GATT_ILLEGAL_UUID (C macro), 314
ESP_GATT_MAX_ATTR_LEN (C macro), 315
ESP_GATT_MAX_READ_MULTI_HANDLES macro, 314
ESP_GATT_PERM_READ (C macro), 314
ESP_GATT_PERM_READ_AUTHORIZATION macro, 314
ESP_GATT_PERM_READ_ENC_MITM (C macro), 314
ESP_GATT_PERM_READ_ENCRYPTED (C macro), 314
esp_gatt_perm_t (C++ type), 315
ESP_GATT_PERM_WRITE (C macro), 314
ESP_GATT_PERM_WRITE_AUTHORIZATION macro, 314
ESP_GATT_PERM_WRITE_ENC_MITM (C macro), 314
ESP_GATT_PERM_WRITE_Encrypted (C macro), 314
ESP_GATT_PERM_WRITE_SIGNED (C macro), 314
ESP_GATT_PERM_WRITE_SIGNED_MITM (C macro), 314
ESP_GATT_PERP_WRITE_CANCEL (C macro), 332
ESP_GATT_PERP_WRITE_EXEC (C macro), 332
esp_gatt_prep_write_type (C++ enum), 315
esp_gatt_prep_write_type::ESP_GATT_PERP_WRITE_CANCEL (C++ enumerator), 315
esp_gatt_prep_write_type::ESP_GATT_PERP_WRITE_EXEC (C++ enumerator), 315
ESP_GATT_RSP_BY_APP (C macro), 315
esp_gatt_rsp_t (C++ union), 304
esp_gatt_rsp_t::attr_value (C++ member), 304
esp_gatt_rsp_t::handle (C++ member), 304

Espressif Systems

Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback
enumerator), 317
esp_gatt_status_t::ESP_GATT_NO_RESOURCES (C macro), 311
esp_gatt_status_t::ESP_GATT_NOT_ENCRYPTED (C macro), 311
esp_gatt_status_t::ESP_GATT_NOT_FOUND (C macro), 311
esp_gatt_status_t::ESP_GATT_NOT_LONG (C macro), 311
esp_gatt_status_t::ESP_GATT_OK (C macro), 315
esp_gatt_status_t::ESP_GATT_OUT_OF_RANGE (C macro), 313
esp_gatt_status_t::ESP_GATT_PENDING (C macro), 317
esp_gatt_status_t::ESP_GATT_PROC_IN_PROGRESS (C macro), 310
esp_gatt_status_t::ESP_GATT_SERVICE_STARTED (C macro), 311
esp_gatt_status_t::ESP_GATT_STACK_RSP (C macro), 317
esp_gatt_status_t::ESP_GATT_WRITE_NOT_PERMIT (C macro), 310
esp_gatt_status_t::ESP_GATT_READ_NOT_PERMIT (C macro), 311
esp_gatt_status_t::ESP_GATT_REQ_NOT_SUPPORTED (C macro), 310
esp_gatt_status_t::ESP_GATT_SERVICE_STARTED (C macro), 311
esp_gatt_status_t::ESP_GATTSTACK_RSP (C macro), 317
esp_gatt_status_t::ESP_GATT_WRITE_NOT_PERMIT (C macro), 310
esp_gatt_status_t::ESP_GATT_WRITE_NOT_PERMIT (C macro), 310
esp_gatt_status_t::ESP_GATT_WRITE_NOT_PERMIT (C macro), 310
ESP_GATT_UUID_ALERTLEVEL (C macro), 312
ESP_GATT_UUID_ALERTNTF_SVC (C macro), 310
ESP_GATT_UUID_ALERTSTATUS (C macro), 312
ESP_GATT_UUID_NOTIFICATION (C macro), 310
ESP_GATT_UUID_BATTERYLEVEL (C macro), 312
ESP_GATT_UUID_BATTERY_SERVICE_SVC (C macro), 310
ESP_GATT_UUID_BATTERY_SERVICE_SVC (C macro), 310
ESP_GATT_UUID_BLOODPRESSURESVC (C macro), 310
ESP_GATT_UUID_BLOODPRESSURESVC (C macro), 310
ESP_GATT_UUID_BODYCOMPOSITION (C macro), 310
ESP_GATT_UUID_BONDMANAGEMENT_SVC (C macro), 311
ESP_GATT_UUID_CHARAGG_FORMAT (C macro), 311
ESP_GATT_UUID_CHAR_CLIENT_CONFIG (C macro), 311
ESP_GATT_UUID_CHAR_DECLARE (C macro), 311
ESP_GATT_UUID_CHARDESCRIPTION (C macro), 311
ESP_GATT_UUID_CHAR_EXT_PROP (C macro), 311
ESP_GATT_UUID_CHAR_PRESENT_FORMAT (C macro), 311
ESP_GATT_UUID_CHAR_SRVR_CONFIG (C macro), 311
ESP_GATT_UUID_CHAR_VALID_RANGE (C macro), 311
ESP_GATT_UUID_CONT_GLUCOSE_MONITOR_SVC (C macro), 311
ESP_GATT_UUID_CSC_FEATURE (C macro), 314
ESP_GATT_UUID_CSC_MEASUREMENT (C macro), 313
ESP_GATT_UUID_CYCLING_SPEED_CADENCE_SVC (C macro), 310
ESP_GATT_UUID_CYCLINGPOWER_SVC (C macro), 310
ESP_GATT_UUID_READNOT_PERMIT (C macro), 310
ESP_GATT_UUID_DEVICEINFO_SVC (C macro), 310
ESP_GATT_UUID_ENV_SENSING_CONFIG_DESCR (C macro), 311
ESP_GATT_UUID_ENV_SENSING_MEASUREMENT_DESCR (C macro), 311
ESP_GATT_UUID_ENV_SENSING_TRIGGER_DESCR (C macro), 311
ESP_GATT_UUID_ENVIRONMENTAL_SENSING_SVC (C macro), 310
ESP_GATT_UUID_EXT_RPTREF_DESCR (C macro), 311
ESP_GATT_UUID_FW_VERSIONSTR (C macro), 312
ESP_GATT_UUID_GAPCENTRAL_ADDR_RESOL (C macro), 312
ESP_GATT_UUID_GAP_DEVICE_NAME (C macro), 312
ESP_GATT_UUID_GATT_SRV_CHGD (C macro), 312
ESP_GATT_UUID_GLUCOSE_SVC (C macro), 310
ESP_GATT_UUID_GM_CONTEXT (C macro), 312
ESP_GATT_UUID_GM_FEATURE (C macro), 312
ESP_GATT_UUID_GM_MEASUREMENT (C macro), 312
ESP_GATT_UUID_HEALTH_THERMOM_SVC (C macro), 310
ESP_GATT_UUID_HEART_RATE_SVC (C macro), 310
ESP_GATT_UUID_HID_BT_KB_INPUT (C macro), 313
ESP_GATT_UUID_HID_BT_KB_OUTPUT (C macro), 313
ESP_GATT_UUID_HID_BT_KB_OUTPUT (C macro), 313
ESP_GATT_UUID_HID_BT_KB_INPUT (C macro), 313
ESP_GATT_UUID_HID_BT_KB_OUTPUT (C macro), 313
ESP_GATT_UUID_HID_BT_KB_INPUT (C macro), 313
ESP_GATT_UUID_HID_BT_MOUSE_INPUT (C
macro), 313
ESP_GATT_UUID_HID_CONTROL_POINT (C
macro), 313
ESP_GATT_UUID_HID_INFORMATION (C macro),
313
ESP_GATT_UUID_HID_PROTO_MODE (C macro),
313
ESP_GATT_UUID_HID_REPORT (C macro), 313
ESP_GATT_UUID_HID_REPORT_MAP (C macro),
313
ESP_GATT_UUID_HID_SVC (C macro), 310
ESP_GATT_UUID_IMMEDIATE_ALERT_SVC (C
macro), 310
ESP_GATT_UUID_INCLUDE_SERVICE (C macro),
311
ESP_GATT_UUID_LINK_LOSS_SVC (C macro),
312
ESP_GATT_UUID_LOCAL_TIME_INFO (C macro),
312
ESP_GATT_UUID_LOCATION_AND_NAVIGATION_SVC
(C macro), 310
ESP_GATT_UUID_MANU_NAME (C macro), 313
ESP_GATT_UUID_MODEL_NUMBER_STR (C
macro), 312
ESP_GATT_UUID_NEXT_DST_CHANGE_SVC (C
macro), 310
ESP_GATT_UUID_NUM_DIGITALS_DESCR (C
macro), 311
ESP_GATT_UUID_NW_STATUS (C macro), 312
ESP_GATT_UUID_NW_TRIGGER (C macro), 312
ESP_GATT_UUID_PNP_ID (C macro), 313
ESP_GATT_UUID_PRI_SERVICE (C macro), 311
ESP_GATT_UUID_REF_TIME_INFO (C macro),
312
ESP_GATT_UUID_REF_TIME_UPDATE_SVC (C
macro), 310
ESP_GATT_UUID_RINGER_CP (C macro), 312
ESP_GATT_UUID_RINGER_SETTING (C macro),
312
ESP_GATT_UUID_RPT_REF_DESCR (C macro),
311
ESP_GATT_UUID_RSC_FEATURE (C macro), 313
ESP_GATT_UUID_RSC_MEASUREMENT (C macro),
313
ESP_GATT_UUID_RUNNING_SPEED_CDENCE_SVC
(C macro), 310
ESP_GATT_UUID_SC_CONTROL_POINT (C
macro), 313
ESP_GATT_UUID_SCAN_INT_WINDOW (C macro),
314
ESP_GATT_UUID_SCAN_PARAMETERS_SVC (C
macro), 310
ESP_GATT_UUID_SCAN_REFRESH (C macro), 314
ESP_GATT_UUID_SEC_SERVICE (C macro), 311
ESP_GATT_UUID_SENSOR_LOCATION (C macro),
313
ESP_GATT_UUID_SERIAL_NUMBER_STR (C
macro), 312
ESP_GATT_UUID_SW_VERSION_STR (C macro),
313
ESP_GATT_UUID_SYSTEM_ID (C macro), 312
ESP_GATT_UUID_TIME_TRIGGER_DESCR (C
macro), 311
ESP_GATT_UUID_TX_POWER_LEVEL (C macro),
312
ESP_GATT_UUID_TX_POWER_SVC (C macro), 310
ESP_GATT_UUID_USER_DATA_SVC (C macro),
311
ESP_GATT_UUID_VALUE_TRIGGER_DESCR (C
macro), 311
esp_gatt_value_t::auth_req (C++ member),
307
esp_gatt_value_t::handle (C++ member),
307
esp_gatt_value_t::len (C++ member), 307
esp_gatt_value_t::offset (C++ member),
307
esp_gatt_value_t::value (C++ member), 307
esp_gatt_write_type_t::ESP_GATT_WRITE_TYPE_NO_RSP
(C++ enumerator), 318
esp_gatt_write_type_t::ESP_GATT_WRITE_TYPE_NO_RSP
(C++ enumerator), 319
esp_gatt_write_type_t::ESP_GATT_WRITE_TYPE_RSP
(C++ enumerator), 350
esp_gattc_cb_event_t::ESP_GATT_ACL_EVT
(C++ enumerator), 350
esp_gattc_cb_event_t::ESP_GATT_ADV_DATA_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_ADV_VSC_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_BTH_SCAN_CFG_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_BTH_SCAN_DIS_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_BTH_SCAN_ENB_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_BTH_SCAN_PARAM_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_BTH_SCAN_RD_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_BTH_SCAN_THR_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_CANCEL_OPEN_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_CFG_MTU_EVT
(C++ enumerator), 351
esp_gattc_cb_event_t::ESP_GATT_CLOSE_EVT

Submit Document Feedback
esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_OK
C++ enum
gesp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_ERROR
C++ enum
gesp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CODE
C++ enum
gesp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_ATME
C++ enum
gesp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CME
C++ enum
gesp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CME
C++ enum
gesp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CME
C++ enum
gesp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CME
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_OK
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_ERROR
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_DISCONNECTED
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_CONNECTING
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_CONNECTED_MSBC
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_CONNECTED
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_OK
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_ERROR
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_OK
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_NO_ANSWER
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_ERR
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_DELAYED
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CME
C++ enum

esp_hf_btrh_status_t::ESP_HF_BTRH_STATUS_REJECTED
C++ enum

esp_hf_btrh_status_t::ESP_HF_BTRH_STATUS_HELD
C++ enum

esp_hf_btrh_cmd_t::ESP_HF_BTRH_CMD_REJECT
C++ enum

esp_hf_btrh_cmd_t::ESP_HF_BTRH_CMD_HOLD
C++ enum

esp_hf_btrh_cmd_t::ESP_HF_BTRH_CMD_ACCEPT
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_DISCONNECTED
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_CONNECTING
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_CONNECTED_MSBC
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_OK
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_NO_ANSWER
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_ERR
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_DELAYED
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CME
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_DISCONNECTED
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_CONNECTING
C++ enum

esp_hf_audio_state_t::ESP_HF_AUDIO_STATE_CONNECTED_MSBC
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_OK
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_NO_ANSWER
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_ERR
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_DELAYED
C++ enum

esp_hf_at_response_code_t::ESP_HF_AT_RESPONSE_CODE_CME
C++ enum
esp_hf_client_send_dtmf (C++ function), 437
esp_hf_client_send_nrec (C++ function), 438
esp_hf_client_send_xapl (C++ function), 437
esp_hf_client_start_voice_recognition (C++ function), 435
esp_hf_client_stop_voice_recognition (C++ function), 435
esp_hf_client_volume_update (C++ function), 435
ESP_HF_CLIENT_XAPL_FEAT_BATTERY_REPORT (C macro), 445
ESP_HF_CLIENT_XAPL_FEAT_DOCKED (C macro), 445
ESP_HF_CLIENT_XAPL_FEAT_RESERVED (C macro), 445
ESP_HF_CLIENT_XAPL_NR_STATUS_REPORT (C macro), 445
esp_hf_cme_err_t (C++ enum), 432
esp_hf_cme_err_t::ESP_HF_CME_AG_FAILURE (C++ enumerator), 432
esp_hf_cme_err_t::ESP_HF_CME_DIAL_STRING_TOO_LONG (C++ enumerator), 434
esp_hf_cme_err_t::ESP_HF_CME_INCOMPLETE_PHONE_NUMBER (C++ enumerator), 434
esp_hf_cme_err_t::ESP_HF_CME_INCOMPLETE_SUBSCRIPTION (C++ enumerator), 434
esp_hf_cme_err_t::ESP_HF_CME_MEMORY_FAILURE (C++ enumerator), 433
esp_hf_cme_err_t::ESP_HF_CME_MEMORY_FULL (C++ enumerator), 433
esp_hf_cme_err_t::ESP_HF_CME_NETWORK_NOT_ALLOWED (C++ enumerator), 434
esp_hf_cme_err_t::ESP_HF_CME_NETWORK_TIMEOUT (C++ enumerator), 434
esp_hf_cme_err_t::ESP_HF_CME_NO_CONNECTION_TO_PHONE (C++ enumerator), 432
esp_hf_cme_err_t::ESP_HF_CME_OPERATION_NOT_ALLOWED (C++ enumerator), 433
esp_hf_cme_err_t::ESP_HF_CME_OPERATION_NOT_SUPPORTED (C++ enumerator), 433
esp_hf_cme_err_t::ESP_HF_CME_PH_SIM_PIN_REQUIRED (C++ enumerator), 433
esp_hf_cme_err_t::ESP_HF_CME_SIM_BUSY (C++ enumerator), 433
esp_hf_cme_err_t::ESP_HF_CME_SIM_FAILURE (C++ enumerator), 433
esp_hf_cme_err_t::ESP_HF_CME_SIM_NOT_INSERTED (C++ enumerator), 433

Espresif Systems 2651 Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback
Espresif Systems

2652

Submit Document Feedback
esp_hf_vr_state_t (C++ enum), 426
esp_hf_vr_state_t::ESP_HF_VR_STATE_DISCONNECTED (C++ enum), 426
esp_hf_vr_state_t::ESP_HF_VR_STATE_DISCONNECTING (C++ enum), 426

esp_hf_wbs_config_t (C++ enum), 431
esp_hf_wbs_config_t::ESP_HF_WBS_NO (C++ enum), 431
esp_hf_wbs_config_t::ESP_HF_WBS_NONE (C++ enum), 431
esp_hf_wbs_config_t::ESP_HF_WBS_YES (C++ enum), 431

ESP_HID_CLASS_CDR (C macro), 467
ESP_HID_CLASS_COM (C macro), 467
ESP_HID_CLASS_DGT (C macro), 467
ESP_HID_CLASS_GPD (C macro), 467
ESP_HID_CLASS_JOS (C macro), 467
ESP_HID_CLASS_KBD (C macro), 467
ESP_HID_CLASS_MIC (C macro), 467
ESP_HID_CLASS_RMC (C macro), 467
ESP_HID_CLASS_SED (C macro), 467
ESP_HID_CLASS_UNKNOWN (C macro), 467

esp_hidd_app_param_t (C++ struct), 466
esp_hidd_app_param_t::desc_list (C++ member), 466
esp_hidd_app_param_t::desc_list_len (C++ member), 466
esp_hidd_app_param_t::description (C++ member), 466
esp_hidd_app_param_t::name (C++ member), 466
esp_hidd_app_param_t::provider (C++ member), 466
esp_hidd_app_param_t::subclass (C++ member), 466

esp_hidd_boot_report_id_t (C++ enum), 469
esp_hidd_boot_report_id_t::ESP_HIDD_BOOT_REPORT_ID_MOUSE (C++ enum), 469
esp_hidd_boot_report_id_t::ESP_HIDD_BOOT_REPORT_ID_KEYBOARD (C++ enum), 469

esp_hidd_boot_report_id_t::ESP_HIDD_BOOT_REPORT_ID_MOUSE (C++ struct), 469

esp_hidd_boot_report_id_t::ESP_HIDD_BOOT_REPORT_ID_KEYBOARD (C++ struct), 469

C++ struct
ego.
esp_http_client_auth_type_t::HTTP_AUTH_TYPE_NONE
esp_http_client_auth_type_t::HTTP_AUTH_TYPE_DIGEST
esp_http_client_auth_type_t::HTTP_AUTH_TYPE_BASIC
(esp_http_client_auth_type_t)
esp_http_client_add_auth
(esp_himem_unmap
(esp_himem_rangehandle_t
(ESP_HIMEM_MAPFLAG_RO
(esp_himem_map
(esp_himem_handle_t
esp_himem_get_phys_size
(esp_himem_get_free_size
esp_himem_free_map_range
(ESP_HIMEM_BLKSZ
(ESP_HiddStatus_t
esp_hidd_status_t::ESP_HIDD_SUCCESS
esp_hidd_status_t::ESP_HIDD_NO_RES
esp_hidd_status_t::ESP_HIDD_NO_DATA
esp_hidd_status_t::ESP_HIDD_NEED_REG
esp_hidd_status_t::ESP_HIDD_NEED_DEREG
esp_hidd_status_t::ESP_HIDD_NEED_INIT
esp_hidd_status_t::ESP_HIDD_NEED_DEREG
esp_hidd_status_t::ESP_HIDD_NEED_REG
esp_hidd_status_t::ESP_HIDD_NO_CONNECTION
esp_hidd_status_t::ESP_HIDD_NO_DATA
esp_hidd_status_t::ESP_HIDD_NO_RES
esp_hidd_status_t::ESP_HIDD_ERROR
(esp_himem_alloc
(esp_himem_alloc_map_range
(ESP_HIMEM_BLKSZ
esp_himem_free
(esp_himem_map_range
esp_himem_get_free_size
esp_himem_get_phys_size
esp_himem_handle_t
esp_himem_map
ESP_HIMEM_MAPFLAG_RO
esp_himem_rangehandle_t
esp_himem_reserved_area_size
esp_himem_unmap
esp_http_client_add_auth
(esp_http_client_config_t
esp_http_client_cleanup
esp_http_client_close
esp_http_client_config_t::auth_type
esp_http_client_config_t::buffer_size
esp_http_client_config_t::buffer_size_tx
esp_http_client_config_t::cert_len
esp_http_client_config_t::cert_pem
esp_http_client_config_t::client_cert_len
esp_http_client_config_t::client_cert_pem
esp_http_client_config_t::client_key_len
esp_http_client_config_t::client_key_password
esp_http_client_config_t::client_key_password_len
esp_http_client_config_t::client_key_pem
esp_http_client_config_t::common_name
esp_http_client_config_t::crt_bundle_attach
esp_http_client_config_t::event_handler
esp_http_client_config_t::host
esp_http_client_config_t::if_name
esp_http_client_config_t::is_async
esp_http_client_config_t::keep_alive_count
esp_http_client_config_t::keep_alive_enable
esp_http_client_config_t::keep_alive_idle
esp_http_client_config_t::keep_alive_interval
esp_http_client_config_t::max_authorization_retries
esp_http_client_config_t::max_redirection_count
esp_http_client_config_t::max_redirection_count
esp_http_client_config_t::max_redirection_count
esp_http_client_config_t::method
esp_http_client_config_t::password
esp_http_client_config_t::path
esp_http_client_config_t::port
esp_http_client_event_t (C++ type), 168
esp_http_client_fetch_headers (C++ function), 162
esp_http_client_flush_response (C++ function), 164
esp_http_client_get_chunk_length (C++ function), 165
esp_http_client_get_content_length (C++ function), 163
esp_http_client_get_errno (C++ function), 161
esp_http_client_get_header (C++ function), 161
esp_http_client_get_password (C++ function), 160
esp_http_client_get_post_field (C++ function), 160
esp_http_client_get_status_code (C++ function), 163
esp_http_client_get_transport_type (C++ function), 163
esp_http_client_get_url (C++ function), 164
esp_http_client_get_username (C++ function), 160
esp_http_client_handle_t (C++ type), 168
esp_http_client_init (C++ function), 159
esp_http_client_is_chunked_response (C++ function), 162
esp_http_client_is_complete_data_received (C++ function), 164
esp_http_client_method_t (C++ enum), 170
esp_http_client_method_t::HTTP_METHOD_COPY (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_DELETE (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_GET (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_HEAD (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_LOCK (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_MAX (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_MKCOL (C++ enumerator), 171
esp_http_client_method_t::HTTP_METHOD_MOVE (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_NOTIFY (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_OPTIONS (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_PATCH (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_PROPFIND (C++ enumerator), 170
esp_http_client_method_t::HTTP_METHOD_PROPPATCH (C++ enumerator), 170

esp_http_client_event_t::HTTP_EVENT_REDIRECT (C++ member), 165
esp_http_client_event_t::HTTP_EVENT_ON_HEADER (C++ member), 166
esp_http_client_event_t::HTTP_EVENT_ON_FINISH (C++ member), 166
esp_http_client_event_t::HTTP_EVENT_ON_DATA (C++ member), 167
esp_http_client_event_t::HTTP_EVENT_ON_CONNECTED (C++ member), 167
esp_http_client_event_t::HTTP_EVENT_HEADERS_SENT (C++ member), 167
esp_http_client_event_t::HTTP_EVENT_HEADER_SENT (C++ member), 167
esp_http_client_event_t::HTTP_EVENT_ERROR (C++ member), 165
esp_http_client_event_t::HTTP_EVENT_DISCONNECTED (C++ member), 165

esp_http_client_config_t::use_global_ca_store (C++ member), 167
esp_http_client_config_t::username (C++ member), 166
esp_http_client_config_t::user_data (C++ member), 167
esp_http_client_config_t::user_agent (C++ member), 166
esp_http_client_config_t::timeout_ms (C++ member), 166
esp_http_client_config_t::transport_type (C++ member), 165
esp_http_client_config_t::query (C++ member), 168

esp_http_client_delete_header (C++ member), 162
esp_http_client_event::user_data (C++ member), 165
esp_http_client_event::header_key (C++ member), 165
esp_http_client_event::header_value (C++ member), 165
esp_http_client_event::data_len (C++ member), 165
esp_http_client_event::event_id (C++ member), 165

esp_http_client_event::query (C++ member), 166
esp_http_client_event::skip_cert_verify (C++ member), 167
esp_http_client_event::timeout_ms (C++ member), 166
esp_http_client_event::transport_type (C++ member), 165
esp_http_client_event::url (C++ member), 165
esp_http_client_event::use_global_ca_store (C++ member), 165

release v5.1-dev-2066-g7869f4e151
Esperrif Systems

Submit Document Feedback

2659

Release v5.1-dev-2066-g7869f4e151

ESP_IP4ADDR_INIT (C macro), 953
esp_ip4addr_ntoa (C++ function), 941
ESP_IP4TOADDR (C macro), 953
ESP_IP4TOUINT32 (C macro), 953
esp_ip6_addr (C++ struct), 951
esp_ip6_addr::addr (C++ member), 951
esp_ip6_addr::zone (C++ member), 951
ESP_IP6_ADDR_BLOCK1 (C macro), 952
ESP_IP6_ADDR_BLOCK2 (C macro), 952
ESP_IP6_ADDR_BLOCK3 (C macro), 952
ESP_IP6_ADDR_BLOCK4 (C macro), 952
ESP_IP6_ADDR_BLOCK5 (C macro), 952
ESP_IP6_ADDR_BLOCK6 (C macro), 952
ESP_IP6_ADDR_BLOCK7 (C macro), 952
ESP_IP6_ADDR_BLOCK8 (C macro), 952
esp_ip6_addr_t (C++ type), 953
esp_ip6_addr_type_t (C++ enum), 953
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_GLOBAL (C++ enumerator), 953
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_IPV4_MAPPED (C++ enumerator), 954
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_IPV6 (C++ enumerator), 954
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_UNIQUE_LOCAL (C++ enumerator), 954
esp_ip6_addr_type_t::ESP_IP6_ADDR_IS_UNIQUE_LOCAL (C++ enumerator), 954
ESP_IP6ADDR_INIT (C macro), 953
esp_ip_addr_t (C++ type), 953
ESP_IPADDR_TYPE_ANY (C macro), 953
ESP_IPADDR_TYPE_V4 (C macro), 953
ESP_IPADDR_TYPE_V6 (C macro), 953
esp_ipc_call (C++ function), 1981
esp_ipc_call_blocking (C++ function), 1982
esp_ipc_func_t (C++ type), 1982
esp_ipc_isr_asm_call (C++ function), 1982
esp_ipc_isr_asm_call_blocking (C++ function), 1982
esp_ipc_isr_func_t (C++ type), 1984
esp_ipc_isr_release_other_cpu (C++ function), 1983
esp_ipc_isr_stall_abort (C++ function), 1983
esp_ipc_isr_stall_other_cpu (C++ function), 1983
esp_ipc_isr_stall_pause (C++ function), 1983
esp_ipc_isr_stall_resume (C++ function), 1983
esp_lcd_del_i80_bus (C++ function), 1090
esp_lcd_i2c_bus_handle_t (C++ type), 1094
esp_lcd_i80_bus_config_t (C++ struct), 1093
esp_lcd_i80_bus_config_t::bus_width (C++ member), 1093
esp_lcd_i80_bus_config_t::clk_src (C++ member), 1093
esp_lcd_i80_bus_config_t::data_gpio_nums (C++ member), 1093
esp_lcd_i80_bus_config_t::dc_gpio_num (C++ member), 1093
esp_lcd_i80_bus_config_t::max_transfer_bytes (C++ member), 1093
esp_lcd_i80_bus_config_t::psram_trans_align (C++ member), 1093
esp_lcd_i80_bus_config_t::sram_trans_align (C++ member), 1093
esp_lcd_i80_bus_config_t::wr_gpio_num (C++ member), 1093
esp_lcd_i80_bus_handle_t (C++ type), 1095
esp_lcd_new_i80_bus (C++ function), 1090
esp_lcd_new_panel_io_i2c (C++ function), 1090
esp_lcd_new_panel_io_i80 (C++ function), 1090
esp_lcd_new_panel_io_spi (C++ function)
esp_lcd_new_panel_nt35510 (C++ function),
esp_lcd_new_panel_st7789 (C++ function)
esp_lcd_new_panel_src1306 (C++ function),
esp_lcd_new_panel_st7789 (C++ function),
esp_lcd_panel_del (C++ function), 1095
esp_lcd_panel_dev_config_t (C++ struct), 1098
esp_lcd_panel_dev_config_t::bits_per_pixel (C++ member), 1098
esp_lcd_panel_dev_config_t::color_space (C++ member), 1098
esp_lcd_panel_dev_config_t::flags (C++ member), 1098
esp_lcd_panel_dev_config_t::reset_active_high (C++ member), 1098
esp_lcd_panel_dev_config_t::reset_gpio_num (C++ member), 1098
esp_lcd_panel_dev_config_t::rgb_endian (C++ member), 1098
esp_lcd_panel_dev_config_t::vendor_config (C++ member), 1098
esp_lcd_panel_dev_config_t::vendor_config (C++ member), 1098
esp_lcd_panel_disp_off (C++ function), 1097
esp_lcd_panel_disp_on_off (C++ function), 1097
esp_lcd_panel_draw_bitmap (C++ function), 1095
esp_lcd_panel_handle_t (C++ type), 1098
esp_lcd_panel_init (C++ function), 1095
esp_lcd_panel_invert_color (C++ function), 1096
esp_lcd_panel_io_callbacks_t (C++ struct), 1091
esp_lcd_panel_io_callbacks_t::on_color_trans_done (C++ member), 1091
esp_lcd_panel_io_color_trans_done_cb_t (C++ type), 1095
ESP_LE_AUTH_REQ_SC_ONLY (C macro), 284
ESP_LE_KEY_LCSRK (C macro), 284
ESP_LE_KEY_LENC (C macro), 284
ESP_LE_KEY_LID (C macro), 284
ESP_LE_KEY_LMK (C macro), 284
ESP_LE_KEY_NONE (C macro), 283
ESP_LE_KEY_PCSRK (C macro), 283
ESP_LE_KEY_PENK (C macro), 283
ESP_LE_KEY_PID (C macro), 283
ESP_LE_KEY_PFLK (C macro), 284
esp_light_sleep_start (C++ function), 2046
esp_link_key (C++ type), 239
esp_local_ctrl_add_property (C++ function), 176
esp_local_ctrl_config (C++ struct), 179
esp_local_ctrl_config::handlers (C++ member), 180
esp_local_ctrl_config::max_properties (C++ member), 180
esp_local_ctrl_config::proto_sec (C++ member), 180
esp_local_ctrl_config::transport (C++ member), 179
esp_local_ctrl_config::transport_config (C++ member), 179
esp_local_ctrl_config_t (C++ type), 181
esp_local_ctrl_get_property (C++ function), 176
esp_local_ctrl_get_transport_ble (C++ function), 175
esp_local_ctrl_get_transport_httpd (C++ function), 175
esp_local_ctrl_handlers (C++ struct), 178
esp_local_ctrl_handlers::get_prop_values (C++ member), 178
esp_local_ctrl_handlers::set_prop_values (C++ member), 178
esp_local_ctrl_handlers::usr_ctx (C++ member), 179
esp_local_ctrl_handlers::usr_ctx_free_fn (C++ member), 179
esp_local_ctrl_handlers_t (C++ type), 180
esp_local_ctrl_prop (C++ struct), 177
esp_local_ctrl_prop::ctx (C++ member), 177
esp_local_ctrl_prop::ctx_free_fn (C++ member), 177
esp_local_ctrl_prop::flags (C++ member), 177
esp_local_ctrl_prop::name (C++ member), 177
esp_local_ctrl_prop::size (C++ member), 177
esp_local_ctrl_prop::type (C++ member), 177
esp_local_ctrl_prop_val (C++ struct), 177
ESP_LOG_BUFFER_HEX DUMP (C macro), 1994
ESP_LOG_EARLY_IMPL (C macro), 1995
esp_log early_timestamp (C++ function), 1993
ESP_LOG_LEVEL (C macro), 1995
esp_log_level_get (C++ function), 1992
ESP_LOG_LEVEL_LOCAL (C macro), 1996
esp_log_level_set (C++ function), 1992
esp_log_level_t (C++ enum), 1997
esp_log_level_t::ESP_LOG_DEBUG (C++ enumerator), 1997
esp_log_level_t::ESP_LOG_ERROR (C++ enumerator), 1997
esp_log_level_t::ESP_LOG_INFO (C++ enumerator), 1997
esp_log_level_t::ESP_LOG_NONE (C++ enumerator), 1997
esp_log_level_t::ESP_LOG_VERBOSE (C++ enumerator), 1997
esp_log_level_t::ESP_LOG_WARN (C++ enumerator), 1997
esp_log_set_vprintf (C++ function), 1992
esp_log_system_timestamp (C++ function), 1993
esp_log_timestamp (C++ function), 1993
esp_log_write (C++ function), 1993
esp_log_writev (C++ function), 1993
ESP_LOGD (C macro), 1995
ESP_LOGE (C macro), 1995
ESP_LOGI (C macro), 1995
ESP_LOGV (C macro), 1995
ESP_LOGW (C macro), 1995
esp_mac_addr_len_get (C++ function), 2004
esp_mac_type_t (C++ enum), 2004
esp_mac_type_t::ESP_MAC_BASE (C++ enumerator), 2005
esp_mac_type_t::ESP_MAC_BT (C++ enumerator), 2005
esp_mac_type_t::ESP_MAC_EFUSE_CUSTOM (C++ enumerator), 2005
esp_mac_type_t::ESP_MAC_EFUSE_FACTORY (C++ enumerator), 2005
esp_mac_type_t::ESP_MAC_ETH (C++ enumerator), 2005
esp_mac_type_t::ESP_MAC_IEEE802154 (C++ enumerator), 2005
esp_mac_type_t::ESP_MAC_WIFI_SOFTAP (C++ enumerator), 2005
esp_mac_type_t::ESP_MAC_WIFI_STA (C++ enumerator), 2004
esp_mesh allow_root_conflicts (C++ function), 806
esp_mesh_available_txupQ_num (C++ function), 806
esp_mesh_connect (C++ function), 810
esp_mesh_deinit (C++ function), 797
esp_mesh_delete_group_id (C++ function), 807
esp_mesh_disable_ps (C++ function), 811
esp_mesh_disconnect (C++ function), 810
esp_mesh_enable_ps (C++ function), 811
esp_mesh_fix_root (C++ function), 808
esp_mesh_flush_scan_result (C++ function), 810
esp_mesh_flush_upstream_packets (C++ function), 810
esp_mesh_get_active_duty_cycle (C++ function), 812
esp_mesh_get_ap_assoc_expire (C++ function), 805
esp_mesh_get_ap_authmode (C++ function), 803
esp_mesh_get_ap_connections (C++ function), 803
esp_mesh_get_capacity_num (C++ function), 807
esp_mesh_get_config (C++ function), 801
esp_mesh_get_group_list (C++ function), 807
esp_mesh_get_group_num (C++ function), 807
esp_mesh_get_id (C++ function), 801
esp_mesh_get_ie_crypto_key (C++ function), 808
esp_mesh_get_layer (C++ function), 803
esp_mesh_get_max_layer (C++ function), 802
esp_mesh_get_network_duty_cycle (C++ function), 813
esp_mesh_get_non_mesh_connections (C++ function), 803
esp_mesh_get_parent_bssid (C++ function), 803
esp_mesh_get_root_healing_delay (C++ function), 808
esp_mesh_get_router (C++ function), 801
esp_mesh_get_router_bssid (C++ function), 805
esp_mesh_get_routing_table (C++ function), 805
esp_mesh_get_routing_table_size (C++ function), 805
esp_mesh get_running_active_duty_cycle (C++ function), 813
esp_mesh get_rx_pending (C++ function), 806
esp_mesh get_self_organized (C++ function), 804
esp_mesh get_subnet_nodes_list (C++ function), 810
esp_mesh get_subnet_nodes_num (C++ function), 810
esp_mesh get_topology (C++ function), 811
esp_mesh get_total_node_num (C++ function), 805
esp_mesh get_tsf_time (C++ function), 811
esp_mesh get_tx_pending (C++ function), 806
esp_mesh get_type (C++ function), 802
esp_mesh get vote_percentage (C++ function), 805
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::session_t::last_will::qos
(C++ member), 138
esp_mqtt_client_config_t::credentials_t::session_t::last_will::msg_len
(C++ member), 138
esp_mqtt_client_config_t::credentials_t::session_t::last_will::msg
(C++ member), 138
esp_mqtt_client_config_t::session_t::last_will
(C++ member), 138
esp_mqtt_client_config_t::session_t::keepalive
(C++ member), 138
esp_mqtt_client_config_t::session_t::disable_keepalive
(C++ member), 138
esp_mqtt_client_config_t::session_t::disable_clean_session
(C++ member), 138
esp_mqtt_client_config_t::network_t::timeout_ms
(C++ member), 137
esp_mqtt_client_config_t::network_t::refresh_connection_after_ms
(C++ member), 137
esp_mqtt_client_config_t::network_t::reconnect_timeout_ms
(C++ member), 137
esp_mqtt_client_config_t::network_t::disable_auto_reconnect
(C++ member), 137
esp_mqtt_client_config_t::credentials_t::username
(C++ member), 137
esp_mqtt_client_config_t::credentials_t::set_null_client_id
(C++ member), 137
esp_mqtt_client_config_t::credentials_t::authentication_t::use_secure_element
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::authentication_t::password
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::authentication_t::key_password_len
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::authentication_t::key_len
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::authentication_t::ds_data
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::authentication_t::certificate_len
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::authentication_t::certificate
(C++ member), 136
esp_mqtt_client_config_t::credentials_t::authentication_t
(C++ member), 136
esp_mqtt_client_config_t::session_t::protocol_ver
(C++ member), 137
esp_mqtt_client_config_t::session_t::task
(C++ member), 137
esp_mqtt_client_config_t::session_t::task_t
(C++ struct), 138
esp_mqtt_client_config_t::session_t::task_t::priority
(C++ member), 138
esp_mqtt_client_config_t::session_t::task_t::stack_size
(C++ member), 138
esp_mqtt_client_config_t::session_t::task_t::task_stack_size
(C++ member), 138
esp_mqtt_client_disconnected
(C++ function), 131
esp_mqtt_client_disconnect
(C++ function), 131
esp_mqtt_client_enqueue
(C++ function), 131
esp_mqtt_client_init
(C++ function), 132
esp_mqtt_client_init
(C++ function), 132
esp_mqtt_client_get_outbox_size
(C++ function), 130
esp_mqtt_client_publish
(C++ function), 130
esp_mqtt_client_reconnect
(C++ function), 130
esp_mqtt_client_register_event
(C++ function), 131
esp_mqtt_client_register_event
(C++ function), 131
esp_mqtt_client_register_event
(C++ function), 131
esp_mqtt_client_set_uri
(C++ function), 129
esp_mqtt_client_set_uri
(C++ function), 130
esp_mqtt_client_set_stop
(C++ function), 130
esp_mqtt_client_set_stop
(C++ function), 130
esp_mqtt_client_set_stop
(C++ function), 130
esp_mqtt_client_set_stop
(C++ function), 130
esp_mqtt_client_set_stop
(C++ function), 130
esp_mqtt_client_set_stop
(C++ function), 130
esp_mqtt_client_set_stop
(C++ function), 130
esp_mqtt_connect_return_code_t
(C++ enum), 141
esp_mqtt_connect_return_code_t
(C++ enum), 139
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_connect_return_code_t::MQTT_CONNECTION_ACCEPTED
(C++ enumerator), 141
esp_mqtt_error_codes::connect_return_code
(C++ struct), 132
esp_mqtt_error_code::error_type
(C++ member), 132
esp_mqtt_error_code::error_type
(C++ member), 132
esp_mqtt_event_t::protocol_ver (C++ member), 133
esp_mqtt_event_t::apos (C++ member), 133
esp_mqtt_event_t::aot (C++ member), 133
esp_mqtt_event_t::a_data (C++ member), 133
esp_mqtt_event_t::a_local (C++ member), 133
esp_mqtt_event_t::data (C++ member), 133
esp_mqtt_event_t::data_len (C++ member), 133
esp_mqtt_event_t::eDup (C++ member), 133
esp_mqtt_event_t::error_handle (C++ member), 133
esp_mqtt_event_t::event_id (C++ member), 133
esp_mqtt_event_t::msg_id (C++ member), 133
esp_mqtt_event_t::protocol_ver (C++ member), 133
Espresif Systems 2667

Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback

esp_netif_get_desc (C++ function), 942
esp_netif_get_dns_info (C++ function), 940
esp_netif_get_event_id (C++ function), 942
esp_netif_get_flags (C++ function), 942
esp_netif_get_handle_from_ifkey (C++ function), 942
esp_netif_get_handle_from_netif_impl (C++ function), 958
esp_netif_get_hostname (C++ function), 936
esp_netif_get_ifkey (C++ function), 942
esp_netif_get_io_driver (C++ function), 942
esp_netif_get_ip6_global (C++ function), 941
esp_netif_get_ip6_linklocal (C++ function), 940
esp_netif_get_ip_info (C++ function), 936
esp_netif_get_mac (C++ function), 935
esp_netif_get_netif_impl (C++ function), 958
esp_netif_get_netif_impl_index (C++ function), 937
esp_netif_get_netif_impl_name (C++ function), 937
esp_netif_get_nr_of_ifs (C++ function), 943
esp_netif_get_old_ip_info (C++ function), 936
esp_netif_get_route_prio (C++ function), 942
esp_netif htonl (C macro), 952
esp_netif_inherent_config (C++ struct), 945
esp_netif_inherent_config::bridge_info (C++ member), 946
esp_netif_inherent_config::flags (C++ member), 945
esp_netif_inherent_config::get_ip_event (C++ member), 945
esp_netif_inherent_config::if_desc (C++ member), 945
esp_netif_inherent_config::if_key (C++ member), 945
esp_netif_inherent_config::ip_info (C++ member), 945
esp_netif_inherent_config::lost_ip_event (C++ member), 945
esp_netif_inherent_config::mac (C++ member), 945
esp_netif_inherent_config::route_prio (C++ member), 946
esp_netif_inherent_config_t (C++ type), 948
ESP_NETIF_INHERENT_DEFAULT_OPENTHREAD (C macro), 925
esp_netif_init (C++ function), 932
esp_netif_iodriver_handle (C++ type), 948
esp_netif_ip4_makeu32 (C macro), 952
esp_netif_ip6_get_addr_type (C++ function), 951
esp_netif_ip6_info_t (C++ struct), 943
esp_netif_ip6_info_t::ip (C++ member), 944
esp_netif_ip_addr_copy (C++ function), 951
esp_netif_ip_event_type (C++ enum), 951
esp_netif_ip_event_type::ESP_NETIF_IP_EVENT_GOT_IP (C++ enumerator), 951
esp_netif_ip_event_type::ESP_NETIF_IP_EVENT_GOT_IP (C++ enumerator), 951
esp_netif_ip_event_type_t (C++ type), 948
esp_netif_ip_info_t (C++ struct), 943
esp_netif_ip_info_t::gw (C++ member), 943
esp_netif_ip_info_t::ip (C++ member), 943
esp_netif_ip_info_t::netmask (C++ member), 943
esp_netif_is_netif_up (C++ function), 936
esp_netif_netstack_buf_free (C++ function), 943
esp_netif_netstack_buf_ref (C++ function), 943
esp_netif_netstack_config_t (C++ type), 948
esp_netif_new (C++ function), 932
esp_netif_next (C++ function), 942
esp_netif_pair_mac_ip_t (C++ struct), 947
esp_netif_pair_mac_ip_t::ip (C++ member), 947
esp_netif_pair_mac_ip_t::mac (C++ member), 947
esp_netif_receive (C++ function), 933
esp_netif_receive_t (C++ type), 948
esp_netif_set_default_netif (C++ function), 935
esp_netif_set_dns_info (C++ function), 940
esp_netif_set_driver_config (C++ function), 932
esp_netif_set_hostname (C++ function), 935
esp_netif_set_ip4_addr (C++ function), 941
esp_netif_set_ip_info (C++ function), 936
esp_netif_set_link_speed (C++ function), 958
esp_netif_set_mac (C++ function), 935
esp_netif_set_old_ip_info (C++ function), 937
esp_netif_str_to_ip4 (C++ function), 941
esp_netif_str_to_ip6 (C++ function), 942
esp_netif_t (C++ type), 948
esp_netif_transmit (C++ function), 958
esp_netif_transmit_wrap (C++ function), 959
esp_nimble_hci_deinit (C++ function), 778
esp_nimble_hci_init (C++ function), 777
esp_now_add_peer (C++ function), 788
esp_now_deinit (C++ function), 787
esp_now_del_peer (C++ function), 788
ESP_NOW_ETH_ALEN (C macro), 791
esp_now_fetch_peer (C++ function), 789
esp_now_get_peer (C++ function), 789
esp_now_get_peer_num (C++ function), 789
esp_now_get_version (C++ function), 787
(C++ member), 921
esp_openthread_netif_glue_deinit (C++ function), 925
esp_openthread_netif_glue_init (C++ function), 925
esp_openthread_platform_config_t (C++ struct), 922
esp_openthread_platform_config_t::host_config (C++ member), 922
esp_openthread_platform_config_t::port_config (C++ member), 923
esp_openthread_platform_config_t::radio_config (C++ member), 922
esp_openthread_port_config_t (C++ struct), 922
esp_openthread_port_config_t::netif_queue_size (C++ function), 2021
esp_openthread_port_config_t::storage_partition (C++ member), 922
esp_openthread_port_config_t::task_queue_size (C++ function), 2021
esp_openthread_radio_config_t (C++ struct), 922
esp_openthread_radio_config_t::radio_mode (C++ member), 922
esp_openthread_radio_config_t::radio_uart_config (C++ member), 922
esp_openthread_radio_mode_t (C++ enum), 923
esp_openthread_radio_mode_t::RADIO_MODE_NATIVE (C++ enumerator), 924
esp_openthread_radio_mode_t::RADIO_MODE_SPI_Rx (C++ enumerator), 924
esp_openthread_radio_mode_t::RADIO_MODE_UART_EC (C++ function), 926
esp_openthread_rcc_deinit (C++ function), 926
esp_openthread_rcc_failure_handler (C++ type), 923
esp_openthread_register_rcc_failure_handler (C++ function), 926
esp_openthread_set_backbone_netif (C++ function), 925
esp_openthread_uart_config_t (C++ struct), 921
esp_openthread_uart_config_t::port (C++ member), 921
esp_openthread_uart_config_t::rx_pin (C++ member), 921
esp_openthread_uart_config_t::tx_pin (C++ member), 921
esp_openthread_uart_config_t::uart_config (C++ member), 921
esp_ota_abort (C++ function), 2020
esp_ota_begin (C++ function), 2018
esp_ota_check_rollback_is_possible (C++ function), 2022
esp_ota_end (C++ function), 2019
esp_ota_erase_last_boot_app_partition (C++ function), 2022
esp_ota_get_app_description (C++ function), 2017
esp_ota_get_app_elf_sha256 (C++ function), 2018
esp_ota_get_app_partition_count (C++ function), 2021
esp_ota_get_boot_partition (C++ function), 2020
esp_ota_get_last_invalid_partition (C++ function), 2021
esp_ota_get_next_update_partition (C++ function), 2020
esp_ota_get_partition_description (C++ function), 2021
esp_ota_get_running_partition (C++ function), 2020
esp_ota_get_state_partition (C++ function), 2021
esp_ota_handle_t (C++ type), 2023
esp_ota_mark_app_invalid_rollback_and_reboot (C++ function), 2021
esp_ota_mark_app_valid_cancel_rollback (C++ function), 2021
esp_ota_mark_partition_invalid (C++ function), 2020
esp_ota_write (C++ function), 2018
esp_ota_write_with_offset (C++ function), 2019
esp_partition_check_identity (C++ function), 1719
esp_partition_deregister_external (C++ function), 1720
esp_partition_erase_range (C++ function), 1718
esp_partition_find (C++ function), 1716
esp_partition_find_first (C++ function), 1716
esp_partition_find_next (C++ function), 1716
esp_partition_get (C++ function), 1716
esp_partition_get_sha256 (C++ function), 1719
esp_partition_iterator_release (C++ function), 1716
esp_partition_iterator_t (C++ type), 1721
esp_partition_mmap (C++ function), 1718
esp_partition_mmap_handle_t (C++ type), 1721
esp_partition_mmap_memory_t (C++ enum), 1721
esp_partition_mmap_memory_t::ESP_PARTITION_MMAP_DATA (C++ enumerator), 1721
esp_partition_mmap_memory_t::ESP_PARTITION_MMAP_INST (C++ enumerator), 1721
esp_partition_next (C++ function), 1716
esp_partition_read (C++ function), 1717
esp_partition_read_raw (C++ function), 1717
esp_partition_register_external (C++ function), 1720
ESP_PARTITION_SUBTYPE_OTA (C macro), 1721
esp_partition_subtype_t (C++ enum), 1722
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_NVS (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_FAT (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_ESPHTTPD (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_EFUSE_EM (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_DATA_COREDUMP (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_MIN (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_MAX (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_9 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_8 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_7 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_5 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_4 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_2 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_15 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_14 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_12 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_11 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_10 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_1 (C++ enumerator), 1724
(esp_partition_subtype_t::ESP_PARTITION_SUBTYPE_APP_OTA_0 (C++ enumerator), 1724

---

Submit Document Feedback
esp_spp_cb_param_t::spp_data_ind_evt_param
esp_spp_cb_param_t::spp_start_evt_param::sec_id
(C++ struct), 419
esp_spp_cb_param_t::spp_data_ind_evt_param
esp_spp_cb_param_t::spp_start_evt_param::status
(C++ member), 422
esp_spp_cb_param_t::spp_start_evt_param::use_co
(C++ member), 421
esp_spp_cb_param_t::spp_start_evt_param::handle
(C++ member), 420
esp_spp_cb_param_t::spp_srv_stop_evt_param::status
(C++ member), 422
esp_spp_cb_param_t::spp_srv_stop_evt_param::scn
(C++ member), 419
esp_spp_cb_param_t::spp_srv_stop_evt_param::handle
(C++ member), 421
esp_spp_cb_param_t::spp_srv_stop_evt_param
esp_spp_cb_param_t::spp_srv_open_evt_param::status
(C++ member), 422
esp_spp_cb_param_t::spp_srv_open_evt_param::rem_bda
(C++ member), 419
esp_spp_cb_param_t::spp_srv_open_evt_param::new_listen_handle
(C++ member), 422
esp_spp_cb_param_t::spp_srv_open_evt_param::handle
(C++ member), 420
esp_spp_cb_param_t::spp_srv_open_evt_param
esp_spp_cb_param_t::spp_open_evt_param::status
(C++ member), 422
esp_spp_cb_param_t::spp_open_evt_param::rem_bda
(C++ member), 420
esp_spp_cb_param_t::spp_open_evt_param
esp_spp_cb_param_t::spp_init_evt_param::status
(C++ member), 422
esp_spp_cb_param_t::spp_init_evt_param
esp_spp_cb_param_t::spp_discovery_comp_evt_param::status
(C++ member), 422
esp_spp_cb_param_t::spp_discovery_comp_evt_param::service_name
(C++ member), 421
esp_spp_cb_param_t::spp_discovery_comp_evt_param::scn
(C++ member), 420
esp_spp_cb_param_t::spp_discovery_comp_evt_param
esp_spp_cb_param_t::spp_data_ind_evt_param::status
(C++ member), 422
esp_spp_cb_param_t::spp_data_ind_evt_param::len
(C++ member), 423
esp_spp_cb_param_t::spp_data_ind_evt_param::handle
(C++ member), 421

ESP_SPP_SEC_AUTHENTICATE (C macro), 423
ESP_SPP_SEC_AUTHORIZE (C macro), 423
ESP_SPP_SEC_ENCRYPT (C macro), 423
ESP_SPP_SEC_IN_16_DIGITS (C macro), 423
ESP_SPP_SEC_MIIM (C macro), 423
ESP_SPP_SEC_MODE4_LEVEL4 (C macro), 423
ESP_SPP_SEC_NONE (C macro), 423
esp_spp_sec_t (C++ type), 423
esp_spp_start_discovery (C++ function), 416
esp_spp_start_srv (C++ function), 416
esp_spp_status_t (C++ enum), 424
esp_spp_status_t::ESP_SPP_BUSY (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_FAILURE (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_NEED_DEINIT (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_NEED_INIT (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_NO_CONNECTION (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_NO_DATA (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_NO_RESOURCE (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_NO_SERVER (C++ enumerator), 424
esp_spp_status_t::ESP_SPP_SUCCESS (C++ enumerator), 424
esp_stop_srv (C++ function), 417
esp_stop_srv_scn (C++ function), 417
esp_sysview_vprintf (C++ function), 2102
esp_spp_write (C++ function), 417
esp_spp_dpp_bootstrap_gen (C++ function), 886
esp_spp_dpp_bootstrap_t (C++ type), 887
esp_spp_dpp_deinit (C++ function), 886
esp_spp_dpp_event_cb_t (C++ type), 887
esp_spp_dpp_event_t (C++ enum), 888
esp_spp_dpp_event_t::ESP_SPP_DPP_CFG_RECV (C++ enumerator), 888
esp_spp_dpp_event_t::ESP_SPP_DPP_FAILURE (C++ enumerator), 888
esp_spp_dpp_event_t::ESP_SPP_DPP_URI (C++ enumerator), 888
esp_spp_dpp_init (C++ function), 886
esp_spp_dpp_start_listen (C++ function), 887
esp_spp_dpp_stop_listen (C++ function), 887
esystem_abort (C++ function), 2001
esysview_flush (C++ function), 1759
sysview_heap_trace_alloc (C++ function), 1760
sysview_heap_trace_free (C++ function), 1760
sysview_heap_trace_start (C++ function), 1760
esp_sysview_heap_trace_stop (C++ function), 1760
esp_sysview_vprintf (C++ function), 1759
task_wdt_add (C++ function), 2101
task_wdt_add_user (C++ function), 2101
task_wdt_config_t (C++ struct), 2103
task_wdt_config_t::idle_core_mask (C++ member), 2103
task_wdt_config_t::timeout_ms (C++ member), 2103
task_wdt_deinit (C++ function), 2101
task_wdt_delete (C++ function), 2102
task_wdt_delete_user (C++ function), 2102
task_wdt_init (C++ function), 2100
task_wdt_isr_user_handler (C++ function), 2102
task_wdt_reconfigure (C++ function), 2101
task_wdt_reset (C++ function), 2102
task_wdt_reset_user (C++ function), 2102
task_wdt_status (C++ function), 2102
task_wdt_user_handle_t (C++ type), 2103
timer_cb_t (C++ type), 1977
timer_create (C++ function), 1973
timer_create_args_t (C++ struct), 1977
timer_create_args_t::arg (C++ member), 1977
timer_create_args_t::callback (C++ member), 1977
timer_create_args_t::dispatch_method (C++ member), 1977
timer_create_args_t::name (C++ member), 1977
timer_create_args_t::skip_unhandled_events (C++ member), 1977
timer_deinit (C++ function), 1973
timer_delete (C++ function), 1975
timer_dispatch_t (C++ type), 1977
timer_dispatch_t::ESP_TIMER_MAX (C++ enumerator), 1977
timer_dispatch_t::ESP_TIMER_TASK (C++ enumerator), 1977
timer_dump (C++ function), 1976
timer_early_init (C++ function), 1973
timer_get_expiry_time (C++ function), 1975
timer_get_next_alarm (C++ function), 1975
timer_get_next_alarm_for_wake_up (C++ function), 1975
timer_get_period (C++ function), 1975
timer_get_time (C++ function), 1975
timer_handle_t (C++ type), 1977
esp_timer_init (C++ function), 1973
esp_timer_isr_dispatch_need_yield (C++ function), 1976
esp_timer_new_etime_alarm_event (C++ function), 1976
esp_timer_restart (C++ function), 1974
esp_timer_start_once (C++ function), 1974
esp_timer_start_periodic (C++ function), 1974
esp_timer_stop (C++ function), 1974
esp_tls_cfg (C++ struct), 150
esp_tls_cfg::alpn_protos (C++ member), 150
esp_tls_cfg::cacert_protos (C++ member), 151
esp_tls_cfg::cacert_bytes (C++ member), 151
esp_tls_cfg::cacert_pem_bytes (C++ member), 151
esp_tls_cfg::cacert_pem_bytes (C++ member), 151
esp_tls_cfg::clientcert_bytes (C++ member), 151
esp_tls_cfg::clientcert_pem_buf (C++ member), 151
esp_tls_cfg::clientcert_pem_bytes (C++ member), 151
esp_tls_cfg::clientkey_buf (C++ member), 151
esp_tls_cfg::clientkey_bytes (C++ member), 151
esp_tls_cfg::clientkey_password (C++ member), 151
esp_tls_cfg::clientkey_password_len (C++ member), 151
esp_tls_cfg::clientkey_pem_buf (C++ member), 151
esp_tls_cfg::clientkey_pem_bytes (C++ member), 151
esp_tls_cfg::common_name (C++ member), 152
esp_tls_cfg::crt_bundle_attach (C++ member), 152
esp_tls_cfg::ds_data (C++ member), 152
esp_tls_cfg::if_name (C++ member), 152
esp_tls_cfg::is_plain_tcp (C++ member), 152
esp_tls_cfg::keep_alive_cfg (C++ member), 152
esp_tls_cfg::non_block (C++ member), 151
esp_tls_cfg::psk_hint_key (C++ member), 152
esp_tls_cfg::skip_common_name (C++ member), 152
esp_tls_cfg::timeout_ms (C++ member), 152
esp_tls_cfg::use_global_ca_store (C++ member), 152
esp_tls_cfg::use_secure_element (C++ member), 152
esp_tlsCfg_t (C++ type), 153
esp_tls_conn_destroy (C++ function), 147
esp_tls_conn_http_new (C++ function), 145
esp_tls_conn_http_new_async (C++ function), 146
esp_tls_conn_http_new_sync (C++ function), 146
esp_tls_conn_new_async (C++ function), 146
esp_tls_conn_new_sync (C++ function), 146
esp_tls_conn_read (C++ function), 147
esp_tls_conn_state (C++ enum), 153
esp_tls_conn_state::ESP_TLS_CONNECTING (C++ enumerator), 153
esp_tls_conn_state::ESP_TLS_DONE (C++ enumerator), 153
esp_tls_conn_state::ESP_TLS_FAIL (C++ enumerator), 153
esp_tls_conn_state::ESP_TLS_HANDSHAKE (C++ enumerator), 153
esp_tls_conn_state::ESP_TLS_INIT (C++ enumerator), 153
esp_tls_conn_state_t (C++ type), 152
esp_tls_conn_write (C++ function), 147
ESP_TLS_ERR_SSL_TIMEOUT (C++ macro), 156
ESP_TLS_ERR_SSL_WANT_READ (C++ macro), 156
ESP_TLS_ERR_SSL_WANT_WRITE (C++ macro), 156
esp_tls_error_handle_t (C++ type), 156
esp_tls_error_type (C++ type), 156
esp_tls_error_type_t (C++ type), 156
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_ESP (C++ enumerator), 156
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_MAX (C++ enumerator), 157
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_MBEDTLS (C++ enumerator), 156
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_MBEDTLS_CE (C++ enumerator), 156
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_SYSTEM (C++ enumerator), 156
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_UNKNOWN (C++ enumerator), 156
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_WOLFSSL (C++ enumerator), 156
esp_tls_error_type_t::ESP_TLS_ERR_TYPE_WOLFSSL_CE (C++ enumerator), 156
esp_tls_free_global_ca_store (C++ function), 148
esp_tls_get_and_clear_error_type (C++ function), 148
esp_tls_get_and_clear_last_error (C++ function), 148
esp_tls_get_bytes_avail (C++ function), 147
esp_tls_get_conn_sockfd (C++ function), 147
esp_tls_get_error_type (C++ function), 148
esp_tls_get_global_ca_store (C++ function), 149

esp_vfs_fat_mount_config_t::allocation_unit_size (C++ member), 149
esp_vfs_fat_init (C++ function), 145
esp_vfs_fat_init_global_ca_store (C++ function), 148
dbstorage_last_error (C++ struct), 154
esp_vfs_last_error::esp_vfs_error_code (C++ member), 154
dbstorage_last_error::last_error (C++ member), 154
dbstorage_last_error_t (C++ type), 156
dbstorage_plain_cpx_multi (C++ function), 149
dbstorage_role (C++ enum), 153
dbstorage_role::ESP_TLS_CLIENT (C++ enumera-tor), 153
dbstorage_role::ESP_TLS_SERVER (C++ enumera-tor), 153
esp_vfs_last_error_t (C++ type), 152
dbstorage_set_global_ca_store (C++ function), 148
dbstorage_unregister_shutdown_handler (C++ function), 2000
ESP_UUID_LEN_128 (C macro), 238
ESP_UUID_LEN_16 (C macro), 238
ESP_UUID_LEN_32 (C macro), 238
dbstorage_vendor_ie_cbl_t (C++ type), 853
dbstorage_vfs_close (C++ function), 1734
dbstorage_vfs_dev_uart_port_set_rx_line_endings (C++ function), 1734
dbstorage_vfs_dev_uart_port_set_tx_line_endings (C++ function), 1744
dbstorage_vfs_dev_uart_register (C++ function), 1743
dbstorage_vfs_dev_uart_set_rx_line_endings (C++ function), 1743
dbstorage_vfs_dev_uart_set_tx_line_endings (C++ function), 1743
dbstorage_vfs_dev_uart_use_driver (C++ function), 1744
dbstorage_vfs_dev_uart_use_nonblocking (C++ function), 1744
ESP_VFS_EVENTD_CONFIG_DEFAULT (C macro), 1745
dbstorage_vfs_eventfd_config_t (C++ struct), 1745
dbstorage_vfs_eventfd_config_t::max_fds (C++ member), 1745
dbstorage_vfs_eventfd_register (C++ function), 1745
dbstorage_vfs_eventfd_unregister (C++ function), 1745
dbstorage_vfs_mount_config_t (C++ struct), 1642, 1747
dbstorage_vfs_mount_config_t::allocation_unit_size (C++ member), 1643, 1747
dbstorage_vfs_mount_config_t::disk_status_check_enable (C++ member), 1643, 1747
dbstorage_vfs_mount_config_t::format_if_mount_failed (C++ member), 1642, 1747
dbstorage_vfs_mount_config_t::max_files (C++ member), 1643, 1747
dbstorage_vfs_register (C++ function), 1640
dbstorage_vfs_sdcard_unmount (C++ function), 1643
dbstorage_vfs_sdmmc_mount (C++ function), 1641
dbstorage_vfs_sdmmcUnmount (C++ function), 1641
dbstorage_vfs_sdspi_mount (C++ function), 1642
dbstorage_vfs_spiflash_mount_ro (C++ function), 1643
dbstorage_vfs_spiflash_mount_rw_wl (C++ function), 1746
dbstorage_vfs_spiflash_unmount (C++ function), 1644
dbstorage_vfs_spiflash_unmount_ro (C++ function), 1644
dbstorage_vfs_spiflash_unmount_rw_wl (C++ function), 1747
dbstorage_vfs_unregister_path (C++ function), 1640
ESP_VFS_FLAG_CONTEXT_PTR (C macro), 1742
ESP_VFS_FLAG_DEFAULT (C macro), 1742
dbstorage_tstat (C++ function), 1734
dbstorage_vfs_id_t (C++ type), 1742
dbstorage_vfs_l2tap_eth_filter (C++ function), 954
dbstorage_vfs_l2tap_intf_register (C++ function), 954
dbstorage_vfs_l2tap_intf_unregister (C++ function), 954
esp_vfs_select_triggered_isr (C++ function), 1736
esp_vfs_spiffs_conf_t (C++ struct), 1730
esp_vfs_spiffs_conf_t::base_path (C++ member), 1730
esp_vfs_spiffs_conf_t::format_if_mount_failed (C++ member), 1730
esp_vfs_spiffs_conf_t::max_files (C++ member), 1730
esp_vfs_spiffs_conf_t::partition_label (C++ member), 1730
esp_vfs_spiffs_register (C++ function), 1728
esp_vfs_spiffs_unregister (C++ function), 1728
esp_vfs_stat (C++ function), 1734
esp_vfs_t (C++ struct), 1737
esp_vfs_t::access (C++ member), 1740
esp_vfs_t::access_p (C++ member), 1740
esp_vfs_t::close (C++ member), 1738
esp_vfs_t::close_p (C++ member), 1738
esp_vfs_t::closedir (C++ member), 1740
esp_vfs_t::closedir_p (C++ member), 1739
esp_vfs_t::end_select (C++ member), 1742
esp_vfs_t::fcntl1 (C++ member), 1740
esp_vfs_t::fcntl1_p (C++ member), 1740
esp_vfs_t::flags (C++ member), 1740
esp_vfs_t::fstat (C++ member), 1738
esp_vfs_t::fstat_p (C++ member), 1738
esp_vfs_t::fsync (C++ member), 1740
esp_vfs_t::fsync_p (C++ member), 1740
esp_vfs_t::ftruncate (C++ member), 1741
esp_vfs_t::ftruncate_p (C++ member), 1740
esp_vfs_t::get_socket_select_semaphore (C++ member), 1742
esp_vfs_t::ioctl1 (C++ member), 1740
esp_vfs_t::ioctl1_p (C++ member), 1740
esp_vfs_t::link (C++ member), 1739
esp_vfs_t::link_p (C++ member), 1738
esp_vfs_t::lseek (C++ member), 1738
esp_vfs_t::lseek_p (C++ member), 1737
esp_vfs_t::mkdir (C++ member), 1740
esp_vfs_t::mkdir_p (C++ member), 1740
esp_vfs_t::open (C++ member), 1738
esp_vfs_t::open_p (C++ member), 1738
esp_vfs_t::opendir (C++ member), 1738
esp_vfs_t::opendir_p (C++ member), 1739
esp_vfs_t::pread (C++ member), 1738
esp_vfs_t::pread_p (C++ member), 1738
esp_vfs_t::pwrite (C++ member), 1738
esp_vfs_t::pwrite_p (C++ member), 1738
esp_vfs_t::read (C++ member), 1738
esp_vfs_t::read_p (C++ member), 1738
esp_vfs_t::readdir (C++ member), 1739
esp_vfs_t::readdir_p (C++ member), 1739
esp_vfs_t::readdir_r (C++ member), 1739
esp_vfs_t::readdir_r_p (C++ member), 1739
esp_vfs_t::rename (C++ member), 1739
esp_vfs_t::rename_p (C++ member), 1739
esp_vfs_t::rmdir (C++ member), 1740
esp_vfs_t::rmdir_p (C++ member), 1740
esp_vfs_t::seekdir (C++ member), 1739
esp_vfs_t::seekdir_p (C++ member), 1739
data_t::socket_select (C++ member), 1742
esp_vfs_t::start_select (C++ member), 1742
esp_vfs_t::stat (C++ member), 1738
esp_vfs_t::stat_p (C++ member), 1738
esp_vfs_t::stop_socket_select (C++ member), 1742
esp_vfs_t::stop_socket_select_isr (C++ member), 1742
esp_vfs_t::tcdrain (C++ member), 1741
esp_vfs_t::tcdrain_p (C++ member), 1741
esp_vfs_t::tcflow (C++ member), 1741
esp_vfs_t::tcflow_p (C++ member), 1741
esp_vfs_t::tcflush (C++ member), 1741
esp_vfs_t::tcflush_p (C++ member), 1741
esp_vfs_t::tcgetattr (C++ member), 1741
esp_vfs_t::tcgetattr_p (C++ member), 1741
esp_vfs_t::tcgetsid (C++ member), 1741
esp_vfs_t::tcgetsid_p (C++ member), 1741
esp_vfs_t::tcsendbreak (C++ member), 1742
esp_vfs_t::tcsendbreak_p (C++ member), 1742
esp_vfs_t::tcsetattr (C++ member), 1741
esp_vfs_t::tcsetattr_p (C++ member), 1741
esp_vfs_t::telldir (C++ member), 1739
esp_vfs_t::telldir_p (C++ member), 1739
esp_vfs_t::truncate (C++ member), 1740
esp_vfs_t::truncate_p (C++ member), 1740
esp_vfs_t::unlink (C++ member), 1739
esp_vfs_t::unlink_p (C++ member), 1739
esp_vfs_t::utime (C++ member), 1741
esp_vfs_t::utime_p (C++ member), 1741
esp_vfs_t::write (C++ member), 1737
esp_vfs_t::write_p (C++ member), 1737
esp_vfs_t::write_char (C++ member), 1737
esp_vfs_unregister (C++ function), 1735
esp_vfs_unregister_fd (C++ function), 1736
esp_vfs_unregister_with_id (C++ function), 1735
esp_vfs_usb_serial_jtag_use_driver (C++ function), 1744
esp_vfs_usb_serial_jtag_use_nonblocking (C++ function), 1744
esp_vfs_utime (C++ function), 1734
esp_vfs_write (C++ function), 1734
esp_vhci_host_callback (C++ struct), 490
esp_vhci_host_callback::notify_host_recv (C++ member), 491
esp_vhci_host_callback::notify_host_send_available (C++ member), 490
esp_vhci_host_callback_t (C++ type), 491
eTaskState::eSuspended (C++ enumerator), 1850
ETH_DEFAULT_CONFIG (C macro), 902
eth_esp32_emac_config_t (C++ struct), 910
eth_esp32_emac_config_t::clock_config (C++ member), 911
eth_esp32_emac_config_t::dma_burst_len (C++ member), 911
eth_esp32_emac_config_t::interface (C++ member), 910
eth_esp32_emac_config_t::smi_mdio_gpio_num (C++ member), 910
eth_esp32_emac_config_t::smi_mdio_gpio_num (C++ member), 910
ETH_3P2_EMAC_DEFAULT_CONFIG (C macro), 911
event_t (C++ enum), 905
event_t::ETHERNET_EVENT_CONNECTED (C++ enumerator), 905
event_t::ETHERNET_EVENT_DISCONNECTED (C++ enumerator), 905
event_t::ETHERNET_EVENT_START (C++ enumerator), 905
event_t::ETHERNET_EVENT_STOP (C++ enumerator), 905
event_t::ETHERNET_EVENT_STOP (C++ union), 905
event_t::ETHERNET_EVENT_STOP (C++ union), 905
event_t::ETHERNET_EVENT_STOP (C++ union), 905
mac_config_t::clock_gpio (C++ member), 905
mac_config_t::clock_mode (C++ member), 905
mac_config_t::clock_mode (C++ member), 905
mac_config_t::clock_mode (C++ member), 905
mac_config_t::clock_mode (C++ member), 905
mac_config_t::mii (C++ member), 905
mac_config_t::mii (C++ member), 906
mac_config_t::rmi (C++ member), 906
mac_config_t (C++ struct), 910
mac_config_t (C++ struct), 910
mac_config_t::flags (C++ member), 910
mac_config_t::flags (C++ member), 910
mac_config_t::flags (C++ member), 910
task_stack_size (C++ member), 910
task_stack_size (C++ member), 910
task_stack_size (C++ member), 910
ETH_MAC_DEFAULT_CONFIG (C macro), 911
ETH_MAC_FLAG_PIN_TO_CORE (C macro), 911
ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE (C macro), 911
phy_autoneg_cmd_t (C++ enum), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_DIS (C++ enumerator), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_DIS (C++ enumerator), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_DIS (C++ enumerator), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_DIS (C++ enumerator), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_EN (C++ enumerator), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_EN (C++ enumerator), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_EN (C++ enumerator), 917
phy_autoneg_cmd_t::ESP_ETH_PHY_AUTONEGO_EN (C++ enumerator), 917
phy_config_t (C++ struct), 916
ephy_config_t::autonego_timeout_ms (C++ member), 916
eth_phy_config_t::phy_addr (C++ member), 916
eth_phy_config_t::reset_gpio_num (C++ member), 916
eth_phy_config_t::reset_timeout_ms (C++ member), 916
ETH_PHY_DEFAULT_CONFIG (C macro), 916
ETS_INTERNAL_INTR_SOURCE_OFF (C macro), 1990
ETS_INTERNAL_PROFILEING_INTR_SOURCE (C macro), 1990
ETS_INTERNAL_SW0_INTR_SOURCE (C macro), 1990
ETS_INTERNAL_SW1_INTR_SOURCE (C macro), 1990
ETS_INTERNAL_TIMER0_INTR_SOURCE (C macro), 1989
ETS_INTERNAL_TIMER1_INTR_SOURCE (C macro), 1990
ETS_INTERNAL_TIMER2_INTR_SOURCE (C macro), 1990
ETS_INTERNAL_UNUSED_INTR_SOURCE (C macro), 1990
EventBits_t (C++ type), 1910
eventfd (C++ function), 1745
EventGroupHandle_t (C++ type), 1910
EXT_ADV_TX_PWR_NO_PREFERENCE (C macro), 292

F
ff_diskio_impl_t (C++ struct), 1644
ff_diskio_impl_t::init (C++ member), 1644
ff_diskio_impl_t::ioctl (C++ member), 1645
ff_diskio_impl_t::read (C++ member), 1644
ff_diskio_impl_t::status (C++ member), 1644
ff_diskio_impl_t::write (C++ member), 1644
ff_diskio_register (C++ function), 1644
ff_diskio_register_raw_partition (C++ function), 1645
ff_diskio_register_sdmmc (C++ function), 1645
ff_diskio_register_wl_partition (C++ function), 1645
filter_cb_t (C++ type), 1273

G
get_phy_version_str (C++ function), 2345
gpio_config (C++ function), 1002
gpio_config_t (C++ struct), 1008
gpio_config_t::intr_type (C++ member), 1009
gpio_config_t::mode (C++ member), 1009
gpio_config_t::pin_bit_mask (C++ member), 1009
gpio_config_t::pull_down_en (C++ member), 1009
gpio_config_t::pull_up_en (C++ member), 1009
gpio_deep_sleep_hold_dis (C++ function), 1007
gpio_deep_sleep_hold_en (C++ function), 1007
gpio_drive_cap_t (C++ enum), 1016
gpio_drive_cap_t::GPIO_DRIVE_CAP_0 (C++ enumerator), 1016
gpio_drive_cap_t::GPIO_DRIVE_CAP_1 (C++ enumerator), 1016
gpio_drive_cap_t::GPIO_DRIVE_CAP_2 (C++ enumerator), 1016
gpio_drive_cap_t::GPIO_DRIVE_CAP_3 (C++ enumerator), 1016
gpio_drive_cap_t::GPIO_DRIVE_DEFAULT (C++ enumerator), 1016
gpio_drive_cap_t::GPIO_DRIVE_MAX (C++ enumerator), 1016
gpio_get_drive_capability (C++ function), 1006
gpio_get_level (C++ function), 1004
gpio_hold_dis (C++ function), 1007
gpio_hold_en (C++ function), 1007
gpio_install_isr_service (C++ function), 1006
gpio_int_type_t (C++ enum), 1014
gpio_int_type_t::GPIO_INTR_ANYEDGE (C++ enumerator), 1014
gpio_int_type_t::GPIO_INTR_DISABLE (C++ enumerator), 1014
gpio_int_type_t::GPIO_INTR_HIGH_LEVEL (C++ enumerator), 1015
gpio_int_type_t::GPIO_INTR_LOW_LEVEL (C++ enumerator), 1015
gpio_int_type_t::GPIO_INTR_MAX (C++ enumerator), 1015
gpio_int_type_t::GPIO_INTR_NEGEDGE (C++ enumerator), 1014
gpio_int_type_t::GPIO_INTR_POSEDGE (C++ enumerator), 1014
gpio_intr_disable (C++ function), 1003
gpio_intr_enable (C++ function), 1003
gpio_iomux_in (C++ function), 1007
gpio_iomux_out (C++ function), 1008
GPIO_IS_VALID_DIGITAL_IO_PAD (C macro), 1009
GPIO_IS_VALID_GPIO (C macro), 1009
GPIO_INFO_OUTPUT_GPIO (C macro), 1009
gpio_isr_handle_t (C++ type), 1009
gpio_isr_handler_add (C++ function), 1006
gpio_isr_handler_remove (C++ function), 1006
gpio_isr_register (C++ function), 1005
gpio_isr_t (C++ type), 1009
gpio_mode_t (C++ enum), 1015
gpio_mode_t::GPIO_MODE_DISABLE (C++ enumerator), 1015
gpio_mode_t::GPIO_MODE_INPUT (C++ enumerator), 1015
gpio_mode_t::GPIO_MODE_INPUT_OUTPUT (C++ enumerator), 1015
gpio_mode_t::GPIO_MODE_OUTPUT (C++ enumerator), 1015
gpio_mode_t::GPIO_MODE_OUTPUT_OD (C++ enumerator), 1015
gpio_num_t (C++ enumerator), 1012
gpio_num_t::GPIO_NUM_0 (C++ enumerator), 1012
gpio_num_t::GPIO_NUM_1 (C++ enumerator), 1012
gpio_num_t::GPIO_NUM_10 (C++ enumerator), 1012
gpio_num_t::GPIO_NUM_11 (C++ enumerator), 1012
gpio_num_t::GPIO_NUM_12 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_13 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_14 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_15 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_16 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_17 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_18 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_19 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_2 (C++ enumerator), 1012
gpio_num_t::GPIO_NUM_20 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_21 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_22 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_23 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_24 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_25 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_26 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_27 (C++ enumerator), 1013
gpio_num_t::GPIO_NUM_28 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_29 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_3 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_30 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_31 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_32 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_33 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_34 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_35 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_36 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_37 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_38 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_39 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_4 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_5 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_6 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_7 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_8 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_9 (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_MAX (C++ enumerator), 1014
gpio_num_t::GPIO_NUM_NC (C++ enumerator), 1012
GPIO_PIN_COUNT (C macro), 1009
GPIO_PIN_REG_0 (C macro), 1009
GPIO_PIN_REG_1 (C macro), 1009
GPIO_PIN_REG_10 (C macro), 1010
GPIO_PIN_REG_11 (C macro), 1010
GPIO_PIN_REG_12 (C macro), 1010
GPIO_PIN_REG_13 (C macro), 1010
GPIO_PIN_REG_14 (C macro), 1010
GPIO_PIN_REG_15 (C macro), 1010
GPIO_PIN_REG_16 (C macro), 1010
GPIO_PIN_REG_17 (C macro), 1010
GPIO_PIN_REG_18 (C macro), 1010
GPIO_PIN_REG_19 (C macro), 1010
GPIO_PIN_REG_2 (C macro), 1009
GPIO_PIN_REG_20 (C macro), 1010
GPIO_PIN_REG_21 (C macro), 1010
GPIO_PIN_REG_22 (C macro), 1010
GPIO_PIN_REG_23 (C macro), 1010
GPIO_PIN_REG_24 (C macro), 1010
GPIO_PIN_REG_25 (C macro), 1010
GPIO_PIN_REG_26 (C macro), 1011
Heap Caps Functions:

- `heap_caps_get_free_size`
- `heap_caps_get_info`
- `heap_caps_free`
- `heap_caps_malloc`
- `heap_caps_malloc_prefer`
- `heap_caps_malloc_extmem_enable`
- `heap_caps_free`
- `heap_caps_get_allocated_size`
- `heap_caps_calloc`
- `heap_caps_calloc_prefer`
- `heap_caps_realloc`
- `heap_caps_realloc_prefer`
- `heap_caps_get_largest_free_block`
- `heap_caps_free_nonos_stack_heaps`
- `heap_caps_get_minimum_free_size`
- `heap_caps_get_total_size`
- `heap_caps_get_heap_info`
- `heap_caps_get_heap_info_all`

GPTimer Functions:

- `gptimer_stop`
- `gptimer_start`
- `gptimer_set_alarm_action`
- `gptimer_register_event_callbacks`
- `gptimer_new_timer`
- `gptimer_new_etm_task`
- `gptimer_new_etm_event`
- `gptimer_get_raw_count`
- `gptimer_get_captured_count`

GPTimer Enumerators:

- `gptimer_etm_task_type_t::GPTIMER_ETM_TASK_CAPTURE`
- `gptimer_etm_task_type_t::GPTIMER_ETM_TASK_RELOAD`
- `gptimer_etm_task_type_t::GPTIMER_ETM_TASK_STOP_COUNT`
- `gptimer_etm_task_type_t::GPTIMER_ETM_TASK_MAX`
httpd_req_recv (C++ function), 205
httpd_req_t (C++ type), 223
httpd_req_to_sockfd (C++ function), 205
httpd_respond (C++ function), 208
httpd_respond_404 (C++ function), 211
httpd_respond_408 (C++ function), 212
httpd_respond_500 (C++ function), 212
httpd_respond_chunk (C++ function), 209
httpd_respond_err (C++ function), 211
httpd_respondstr (C++ function), 209
httpd_send (C++ function), 210
httpd_send_hdr (C++ function), 210
httpd_sent_status (C++ function), 210
HTTPD_RESP_USE_STRLEN (C macro), 223
httpd_send (C++ function), 212
httpd_send_func_t (C++ type), 223
httpd_sess_get_ctx (C++ function), 215
httpd_sess_get_transport_ctx (C++ function), 216
httpd_sess_set_ctx (C++ function), 215
httpd_sess_set_pending_override (C++ function), 204
httpd_sess_recv_override (C++ function), 204
httpd_sess_send_override (C++ function), 204
httpd_sess_set_transport_ctx (C++ function), 216
httpd_sess_trigger_close (C++ function), 216
httpd_sess_update_lru_counter (C++ function), 217
HTTPD_SOCKET_RECV (C macro), 221
HTTPD_SOCKET_RECV_INVALID (C macro), 221
HTTPD_SOCKET_RECV_TIMEOUT (C macro), 221
httpd_socket_recv (C++ function), 213
httpd_socket_send (C++ function), 213
httpd_ssl_config (C++ struct), 228
httpd_ssl_config::alpn_protos (C++ member), 229
httpd_ssl_config::cert_len (C++ member), 228
httpd_ssl_config::cacert_pem (C++ member), 228
httpd_ssl_config::cacert_len (C++ member), 228
httpd_ssl_config::cert_select_cb (C++ member), 229
httpd_ssl_config::httpd (C++ member), 228
httpd_ssl_config::port_insecure (C++ member), 228
httpd_ssl_config::port_secure (C++ member), 228
httpd_ssl_config::prvtkey_len (C++ member), 228
httpd_ssl_config::prvtkey_pem (C++ member), 228
httpd_ssl_config::servercert (C++ member), 228
httpd_ssl_config::servercert_len (C++ member), 228
httpd_ssl_config::session_tickets (C++ member), 228
httpd_ssl_config::ssl_userdata (C++ member), 228
httpd_ssl_config::transport_mode (C++ member), 228
httpd_ssl_config::use_secure_element (C++ member), 228
httpd_ssl_config::user_cb (C++ member), 228
HTTPD_SSL_CONFIG_DEFAULT (C macro), 228
httpd_ssl_config_t (C++ type), 229
httpd_ssl_start (C++ function), 227
httpd_ssl_stop (C++ function), 227
httpd_ssl_transport_mode_t (C++ enum), 229
httpd_ssl_transport_mode_t::HTTPD_SSL_TRANSPORT_INSECURE (C++ enumerator), 229
httpd_ssl_transport_mode_t::HTTPD_SSL_TRANSPORT_SECURE (C++ enumerator), 229
httpd_ssl_user_cb_state_t::HTTPD_SSL_USER_CB_SESS_CLOSE (C++ function), 214
httpd_start (C++ function), 214
httpd_stop (C++ function), 215
HTTPD_TYPE_JSON (C macro), 222
HTTPD_TYPE_OCTET (C macro), 222
HTTPD_TYPE_TEXT (C macro), 222
httpd_unregister_uri (C++ function), 204
httpd_unregister_uri_handler (C++ function), 203
httpd_uri (C++ struct), 221
httpd_uri::handler (C++ member), 221
httpd_uri::method (C++ member), 221
httpd_uri::uri (C++ member), 221
httpd_uri::user_ctx (C++ member), 221
httpd_uri_match_func_t (C++ type), 225
httpd_uri_match wildcard (C++ function), 208
httpd_uri_t (C++ type), 223
httpd_work_fn_t (C++ type), 225
HttpStatus_Code::HttpStatus_MovedPermanently (C++ function), 171
HttpStatus_Code::HttpStatus_BadRequest (C++ function), 171
HttpStatus_Code::HttpStatus_Forbidden (C++ function), 171
HttpStatus_Code::HttpStatus_InternalError (C++ function), 172
HttpStatus_Code::HttpStatus_MOVEDPERMANENTLY (C++ function), 171
i2c_master_read (C++ function), 1043
i2c_master_read_byte (C++ function), 1042
i2c_master_read_from_device (C++ function), 1040
i2c_master_start (C++ function), 1042
i2c_master_stop (C++ function), 1043
i2c_master_write (C++ function), 1042
i2c_master_write_byte (C++ function), 1042
i2c_master_write_read_device (C++ function), 1041
i2c_master_write_to_device (C++ function), 1040
i2c_mode_t (C++ enum), 1050
i2c_mode_t::I2C_MODE_MASTER (C++ enumerator), 1050
i2c_mode_t::I2C_MODE_MAX (C++ enumerator), 1050
i2c_mode_t::I2C_MODE_SLAVE (C++ enumerator), 1050
i2c_param_config (C++ function), 1039
i2c_port_t (C++ enum), 1050
i2c_port_t::I2C_NUM_0 (C++ enumerator), 1050
i2c_port_t::I2C_NUM_1 (C++ enumerator), 1050
i2c_port_t::I2C_NUM_MAX (C++ enumerator), 1050
i2c_reset_rx_fifo (C++ function), 1040
i2c_reset_tx_fifo (C++ function), 1039
i2c_rw_t (C++ enum), 1050
i2c_rw_t::I2C_MASTER_READ (C++ enumerator), 1050
i2c_rw_t::I2C_MASTER_WRITE (C++ enumerator), 1050
I2C_SCLK_SRC_FLAG_AWARE_DFS (C macro), 1048
I2C_SCLK_SRC_FLAG_FOR_NOMAL (C macro), 1048
I2C_SCLK_SRC_FLAG_LIGHT_SLEEP (C macro), 1048
i2c_set_data_mode (C++ function), 1046
i2c_set_data_timing (C++ function), 1045
i2c_set_period (C++ function), 1044
i2c_set_pin (C++ function), 1040
i2c_set_start_timing (C++ function), 1045
i2c_set_stop_timing (C++ function), 1045
i2c_set_timeout (C++ function), 1046
i2c_slave_read_buffer (C++ function), 1044
i2c_slave_write_buffer (C++ function), 1043
i2c_trans_mode_t (C++ enum), 1050
i2c_trans_mode_t::I2C_DATA_MODE_LSB_FIRST (C++ enumerator), 1051
i2c_trans_mode_t::I2C_DATA_MODE_MAX (C++ enumerator), 1051
i2c_trans_mode_t::I2C_DATA_MODE_MSB_FIRST (C++ enumerator), 1051
i2s_chan_config_t (C++ struct), 1080
i2s_chan_config_t::auto_clear (C++ member), 1080
i2s_chan_config_t::dma_desc_num (C++ member), 1080
i2s_chan_config_t::dma_frame_num (C++ member), 1080
i2s_chan_config_t::id (C++ member), 1080
i2s_chan_config_t::role (C++ member), 1080
i2s_chan_handler_t (C++ type), 1082
i2s_chan_info_t (C++ struct), 1081
i2s_chan_info_t::dir (C++ member), 1081
i2s_chan_info_t::id (C++ member), 1081
i2s_chan_info_t::mode (C++ member), 1081
i2s_chan_info_t::pair_chan (C++ member), 1081
i2s_chan_info_t::role (C++ member), 1081
I2S_CHANNEL_DEFAULT_CONFIG (C macro), 1081
i2s_channel_disable (C++ function), 1078
i2s_channel_enable (C++ function), 1078
i2s_channel_get_info (C++ function), 1077
i2s_channel_init_pdm_rx_mode (C++ function), 1069
i2s_channel_init_pdm_tx_mode (C++ function), 1071
i2s_channel_init_std_mode (C++ function), 1065
i2s_channel_read (C++ function), 1079
i2s_channel_reconfig_pdm_rx_clock (C++ function), 1070
i2s_channel_reconfig_pdm_rx_gpio (C++ function), 1070
i2s_channel_reconfig_pdm_rx_slot (C++ function), 1070
i2s_channel_reconfig_pdm_tx_clock (C++ function), 1071
i2s_channel_reconfig_pdm_tx_gpio (C++ function), 1072
i2s_channel_reconfig_pdm_tx_slot (C++ function), 1072
i2s_channel_reconfig_std_clock (C++ function), 1065
i2s_channel_reconfig_std_gpio (C++ function), 1066
i2s_channel_reconfig_std_slot (C++ function), 1066
i2s_channel_register_event_callback (C++ function), 1079
i2s_channel_write (C++ function), 1078
i2s_clock_src_t (C++ type), 1083
i2s_comm_mode_t (C++ enum), 1082
i2s_comm_mode_t::I2S_COMM_MODE_NONE (C++ enumerator), 1082
i2s_comm_mode_t::I2S_COMM_MODE_PDM (C++ enumerator), 1082
i2s_comm_mode_t::I2S_COMM_MODE_STD (C++ enumerator), 1082
i2s_data_bit_width_t (C++ enum), 1084
i2s_data_bit_width_t::I2S_DATA_BIT_WIDTH_16BIT member), 1080
i2s_pdm_rx_config_t::gpio_cfg
(C++ member), 1074
i2s_pdm_rx_config_t::slot_cfg
(C++ member), 1074
i2s_pdm_rx_config_t::slot_cfg (C++ struct), 1073
i2s_pdm_rx_config_t::clk
(C++ member), 1073
i2s_pdm_rx_gpio_config_t::clk_inv
(C++ member), 1073
i2s_pdm_rx_gpio_config_t::dins
(C++ member), 1073
i2s_pdm_rx_gpio_config_t::invert_flags
(C++ member), 1073
i2s_pdm_slot_config_t (C++ struct), 1072
i2s_pdm_slot_config_t::data_bit_width
(C++ member), 1073
i2s_pdm_slot_config_t::slot_bit_width
(C++ member), 1073
i2s_pdm_slot_config_t::slot_mask
(C++ member), 1073
I2S_PDM_SLOT_DEFAULT_CONFIG (C macro), 1076
i2s_pdm_slot_mask_t (C++ enum), 1085
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_RIGHT
(C++ member), 1085
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_LEFT
(C++ member), 1085
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_BOTH
(C++ member), 1085
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_LEFT
(C++ member), 1085
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_RIGHT
(C++ member), 1085
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_BOTH
(C++ member), 1085
i2s_pdm_slot_mask_t::I2S_PDM_SLOT_LEFT
(C++ member), 1085
I2S_PDM_TX_DEFAULT_CONFIG (C macro), 1076
i2s_pdm_tx_config_t (C++ struct), 1075
i2s_pdm_tx_conf...
12tap_ioctl_t::L2TAP_S_DEVICE_DRV_HANDLE (C++ enumerator), 955
12tap_ioctl_t::L2TAP_S_INTF_DEV (C++ enumerator), 955
12tap_ioctl_t::L2TAP_S_RCV_FILTER (C++ enumerator), 955
12tap_iowriter (C++ type), 1086
L2TAP_VFS_CONFIG_DEFAULT (C macro), 954
L2TAP_VFS_CONFIG_DEFAULT_PATH (C macro), 954
L2TAP_VFS_DEFAULT (C macro), 954
lcd_clock_source_t (C++ type), 1086
lcd_color_convert_t::LCD_YUV_CONV_STD_BT709 (C++ enum), 1087
lcd_color_convert_t::LCD_YUV_CONV_STD_BT601 (C++ enum), 1087
lcd_color_convert_t::LCD_COLOR_RANGE_FULL (C++ enum), 1087
lcd_color_convert_t::LCD_COLOR_RANGE_LIMIT (C++ enum), 1087
lcd_color_convert_t::LCD_RGB_ENDIAN_RGB (C++ enum), 1087
lcd_color_convert_t::LCD_RGB_ENDIAN_YBG (C++ enum), 1087
lcd_color_convert_t::LCD_COLOR_SPACE_RGB (C++ enum), 1087
lcd_color_convert_t::LCD_COLOR_SPACE_YUV (C++ enum), 1087
lcd_color_convert_t::LCD_COLOR_RANGE_FULL (C++ struct), 1110
ledc_channel_config_t::channel (C++ member), 1110
ledc_channel_config_t::duty (C++ member), 1110
ledc_channel_config_t::flags (C++ member), 1110
ledc_channel_config_t::gpio_num (C++ member), 1110
ledc_channel_config_t::hpoint (C++ member), 1110
ledc_channel_config_t::intr_type (C++ member), 1110
ledc_channel_config_t::output_invert (C++ member), 1110
ledc_channel_config_t::speed_mode (C++ member), 1110
ledc_channel_config_t::timer_sel (C++ member), 1110
ledc_channel_config_t::timer_sel (C++ function), 1106
ledc_channel config (C++ function), 1102
Release v5.1-dev-2066-g7869f4e151
Submit Document Feedback

Espressif Systems

2693
C++ enumerator, 1113
ledc_duty_direction_t::LEDC_DUTY_DIR_MAX (C++ enumerator), 1113
ledc_error_t (C macro), 1112
LED_C叁D_DUTY (C macro), 1112
ledc_fade_func_install (C++ function), 1107
ledc_fade_func_uninstall (C++ function), 1108
ledc_fade_mode_t (C++ enumerator), 1116
ledc_fade_mode_t::LEDC_FADE_MAX (C++ enumerator), 1116
ledc_fade_mode_t::LEDC_FADE_NO_WAIT (C++ enumerator), 1116
ledc_get_hpoint (C++ function), 1103
ledc_get_freq (C++ function), 1104
ledc_get_hpoint (C++ function), 1104
ledc_intr_type_t (C++ enumerator), 1113
ledc_isr_register (C++ function), 1116
ledc_isr_handle_t (C++ type), 1112
ledc_isr_register (C++ function), 1105
ledc_mode_t (C++ function), 1112
ledc_mode_t::LEDC_MODEMAX (C++ function), 1112
ledc_mode_t::LEDC_MODE_MIN (C++ function), 1112
ledc_mode_t::LEDC_MODE_MAX (C++ function), 1112
ledc_timer_rst (C++ function), 1103
ledc_set_duty_with_time (C++ function), 1107
ledc_set_duty (C++ function), 1108
ledc_set_duty_with_time (C++ function), 1107
ledc_set_duty_with_step (C++ function), 1106
ledc_set_freq (C++ function), 1103
ledc_set_freq (C++ function), 1103
dedc_set_freq (C++ function), 1103
ledc_timer_set (C++ function), 1105
ledc_timer_t (C++ enum), 1114
ledc_timer_t::LEDTC_TIMER_0 (C++ enumerator), 1114
ledc_timer_t::LEDTC_TIMER_1 (C++ enumerator), 1114
ledc_timer_t::LEDTC_TIMER_2 (C++ enumerator), 1114
ledc_timer_t::LEDTC_TIMER_3 (C++ enumerator), 1114
ledc_timer_t::LEDTC_TIMER_MAX (C++ enumerator), 1114
ledc_update_duty (C++ function), 1102
linenoiseCompletions (C++ type), 1773

M
M_BGE (C macro), 2092
M_BL (C macro), 2091
M_BRANCH (C macro), 2091
M_BSEQ (C macro), 2092
M_BGGE (C macro), 2092
M_BSGT (C macro), 2093
M_BSELE (C macro), 2092
M_BSLT (C macro), 2092
M_BX (C macro), 2092
M_BXF (C macro), 2092
M_BXZ (C macro), 2092
M_LABEL (C macro), 2091
M_LABELPC (C macro), 2091
M_MOVL (C macro), 2091
MAC2STR (C macro), 2004
MACSTR (C macro), 2004
MALLOC_CAP_32BIT (C macro), 1954
MALLOC_CAP_8BIT (C macro), 1954
MALLOC_CAP_DEFAULT (C macro), 1954
MALLOC_CAP_DMA (C macro), 1954
MALLOC_CAP_EXEC (C macro), 1954
MALLOC_CAP_INTERNAL (C macro), 1954
MALLOC_CAP_INVALID (C macro), 1955
MALLOC_CAP_IRAM_8BIT (C macro), 1955
MALLOC_CAP_PTRD2 (C macro), 1954
MALLOC_CAP_PTRD3 (C macro), 1954
MALLOC_CAP_PTRD4 (C macro), 1954
MALLOC_CAP_PTRD5 (C macro), 1954
MALLOC_CAP_PTRD6 (C macro), 1954
MALLOC_CAP_PTRD7 (C macro), 1954
MALLOC_CAP_RETENTION (C macro), 1955
MALLOC_CAP_RTCRAM (C macro), 1955
MALLOC_CAP_SPIRAM (C macro), 1954
MAX_BLE_DEVMNAME_LEN (C macro), 1614
MAX_BLE_MANUFACTURER_DATA_LEN (C macro), 1615
MAX_FDS (C macro), 1742
MAX_PASSPHRASE_LEN (C macro), 872
MAX_SSID_LEN (C macro), 872
MAX_WPS_AP_CRED (C macro), 872
mcpwm_brake_config_t (C++ struct), 1142
mcpwm_brake_config_t::brake_mode (C++ member), 1143
mcpwm_brake_config_t::cbc_recover_on_tep (C++ member), 1143
mcpwm_brake_config_t::cbc_recover_on_tez (C++ member), 1143
mcpwm_brake_config_t::fault (C++ member), 1143
mcpwm_brake_config_t::flags (C++ member), 1143
mcpwm_brake_event_cb_t (C++ type), 1161
mcpwm_brake_event_data_t (C++ struct), 1159
mcpwm_cap_channel_handle_t (C++ type), 1160
mcpwm_cap_timer_handle_t (C++ type), 1160
mcpwm_capture_channel_config_t (C++ struct), 1158
mcpwm_capture_channel_config_t::flags (C++ member), 1159
mcpwm_capture_channel_config_t::gpio_num (C++ member), 1158
mcpwm_capture_channel_config_t::invert_cap_signal (C++ member), 1158
mcpwm_capture_channel_config_t::io_loop_back (C++ member), 1159
mcpwm_capture_channel_config_t::neg_edge (C++ member), 1158
mcpwm_capture_channel_config_t::pos_edge (C++ member), 1158
mcpwm_capture_channel_config_t::prescale (C++ member), 1158
mcpwm_capture_channel_config_t::pull_down (C++ member), 1158
mcpwm_capture_channel_config_t::pull_up (C++ member), 1158
mcpwm_capture_channel_disable (C++ function), 1157
mcpwm_capture_channel_enable (C++ function), 1156
mcpwm_capture_channel_register_event_callbacks (C++ function), 1157
mcpwm_capture_channel_trigger_soft_capture (C++ function), 1157
mcpwm_capture_clock_source_t (C++ type), 1161
mcpwm_capture_edge_t (C++ enum), 1163
mcpwm_capture_edge_t::MCPWM_CAP_EDGE_NEG (C++ enumerator), 1163
mcpwm_capture_edge_t::MCPWM_CAP_EDGE_POS (C++ enumerator), 1163
mcpwm_capture_event_callbacks_t (C++ struct), 1159
mcpwm_capture_eventCallbacks_t (C++ type), 1161
mcpwm_capture_event_data_t (C++ struct), 1160
mcpwm_capture_event_data_t::cap_edge
Espressif Systems

nvs_get_i64
nvs_get_i32

nvs_flash_init_partition_ptr
nvs_flash_erase_partition
nvs_flash_erase

nvs_entry_info_t::namespace_name

N

nvs_close
nvs_commit
NVS_DEFAULT_PART_NAME
nvs_entry_find
nvs_entry_info
nvs_entry_info_t::key
nvs_entry_info_t::namespace_name

nvs_entry_info_t::type
nvs_entry_next
nvs_erase_all
nvs_erase_key
nvs_flash_deinit
nvs_flash_deinit_partition
nvs_flash_erase
nvs_flash_erase_partition
nvs_flash_erase_partition_ptr
nvs_flash_generate_keys
nvs_flash_init
nvs_flash_init_partition
nvs_flash_init_partition_ptr
nvs_flash_read_security_cfg
nvs_flash_secure_init
nvs_flash_secure_init_partition
nvs_get_blob
nvs_get_i8
nvs_get_i16
nvs_get_i32
nvs_get_i64
nvs_get_u8
nvs_get_u16
nvs_get_u32
nvs_get_u64
nvs_get_used_entry_count

nvs_open
nvs_open_from_partition

nvs_open_mode
nvs_open_mode_t::NVS_READONLY
nvs_open_mode_t::NVS_READWRITE

nvs_part_name_max_size
nvs_part_name
nvs_part_size

multi_heap_info_t::total_blocks
multi_heap_info_t::largest_free_block
multi_heap_info_t::total_free_bytes
multi_heap_malloc
multi_heap_minimum_free_size
multi_heap_realloc
multi_heap_register
multi_heap_set_lock

name_uuid
name_uuid::name
name_uuid::uuid
nvs_close
nvs_commit
NVS_DEFAULT_PART_NAME
nvs_entry_find
nvs_entry_info
nvs_entry_info_t::key
nvs_entry_info_t::namespace_name

nvs_entry_info_t::type
nvs_entry_next
nvs_erase_all
nvs_erase_key
nvs_flash_deinit
nvs_flash_deinit_partition
nvs_flash_erase
nvs_flash_erase_partition
nvs_flash_erase_partition_ptr
nvs_flash_generate_keys
nvs_flash_init
nvs_flash_init_partition
nvs_flash_init_partition_ptr
nvs_flash_read_security_cfg
nvs_flash_secure_init
nvs_flash_secure_init_partition
nvs_get_blob
nvs_get_i8
nvs_get_i16
nvs_get_i32
nvs_get_i64
nvs_get_u8
nvs_get_u16
nvs_get_u32
nvs_get_u64
nvs_get_used_entry_count

nvs_open
nvs_open_from_partition

nvs_open_mode
nvs_open_mode_t::NVS_READONLY
nvs_open_mode_t::NVS_READWRITE

nvs_part_name_max_size
nvs_part_name
nvs_part_size
protocomm_security_t (C++ type), 1611
protocomm_set_security (C++ function), 1608
protocomm_set_version (C++ function), 1608
protocomm_t (C++ type), 1609
protocomm_transport_ble_event_t (C++ enum), 1615
protocomm_transport_ble_event_t:PROTOCOL BLE DISCONNECTED (C++ enumerator), 1615
protocomm_transport_ble_event_t:PROTOCOL BLE CONNECTED (C++ enumerator), 1615
protocomm_unset_security (C++ function), 1608
protocomm_unset_version (C++ function), 1609
PROV_DATA_FLAGS_FLAG (C macro), 541
PROV_DATA_IV_INDEX_FLAG (C macro), 541
PROV_DATA_NET_IDX_FLAG (C macro), 541
psk_key_hint (C++ type), 152
psk_key_hint::hint (C++ member), 150
psk_key_hint::key (C++ member), 150
psk_key_hint::key_size (C++ member), 150
PTHREAD_STACK_MIN (C macro), 2036
pvTaskGetThreadLocalStoragePointer (C++ function), 1834
pxTimerGetTimerID (C++ function), 1888
pxTaskGetTaskStart (C++ function), 1834
Q
QueueHandle_t (C++ type), 1869
QueueSetHandle_t (C++ type), 1869
QueueSetMemberHandle_t (C++ type), 1869
R
R0 (C macro), 2085
R1 (C macro), 2085
R2 (C macro), 2085
R3 (C macro), 2085
RD_REG_PERIPH_RTC_CNTL (C macro), 2085
RD_REG_PERIPH_RTC_I2C (C macro), 2086
RD_REG_PERIPH_RTC_IO (C macro), 2085
RD_REG_PERIPH_SENS (C macro), 2086
Ringbuffer_t (C++ enum), 1944
RingbufferType_t:RINGBUF_TYPE_ALLOWSPLIT (C++ enumerator), 1944
RingbufferType_t:RINGBUF_TYPE_BYTEBUF (C++ enumerator), 1944
RingbufferType_t:RINGBUF_TYPE_MAX (C++ enumerator), 1945
RingbufferType_t:RINGBUF_TYPE_NOSPLIT (C++ enumerator), 1944
RingbufHandle_t (C++ type), 1944
rmt_apply_carrier (C++ function), 1197
rmt_bytes_encoder_config_t (C++ struct), 1199
rmt_bytes_encoder_config_t::bit0 (C++ member), 1200
rmt_bytes_encoder_config_t::bit1 (C++ member), 1200
rmt_bytes_encoder_config_t::flags (C++ member), 1200
rmt_bytes_encoder_config_t::msb_first (C++ member), 1200
rmt_carrier_config_t::always_on (C++ member), 1197
rmt_carrier_config_t::duty_cycle (C++ member), 1198
rmt_carrier_config_t::flags (C++ member), 1198
rmt_carrier_config_t::frequency_hz (C++ member), 1198
rmt_carrier_config_t::polarity_active_low (C++ member), 1198
rmt_channel_handle_t (C++ type), 1201
rmt_clock_source_t (C++ type), 1202
rmt_copy_encoder_config_t (C++ struct), 1200
rmt_del_channel (C++ function), 1197
rmt_del_encoder (C++ function), 1198
rmt_del_sync_manager (C++ function), 1192
rmt_disable (C++ function), 1197
rmt_enable (C++ function), 1197
rmt_encode_state_t (C++ enum), 1200
rmt_encode_state_t::RMT_ENCODING_COMPLETE (C++ enumerator), 1200
rmt_encode_state_t::RMT_ENCODING_MEM_FULL (C++ enumerator), 1200
rmt_encoder_handle_t (C++ type), 1201
rmt_encoder_reset (C++ function), 1199
rmt_encoder_t (C++ struct), 1199
rmt_encoder_t (C++ type), 1200
rmt_encoder_t::del (C++ member), 1199
rmt_encoder_t::encode (C++ member), 1199
rmt_encoder_t::reset (C++ member), 1199
rmt_new_bytes_encoder (C++ function), 1198
rmt_new_copy_encoder (C++ function), 1198
rmt_new_rx_channel (C++ function), 1194
rmt_new_sync_manager (C++ function), 1192
rmt_new_tx_channel (C++ function), 1190
rmt_receive (C++ function), 1194
rmt_receive_config_t (C++ struct), 1196
rmt_receive_config_t::signal_range_max_ns (C++ member), 1196
rmt_receive_config_t::signal_range_min_ns (C++ member), 1196
rmt_rx_channel_config_t (C++ struct), 1196
rmt_rx_channel_config_t::clk_src (C++ member), 1196
rmt_rx_channel_config_t::flags (C++ member), 1196
rmt_rx_channel_config_t::gpio_num (C++ member), 1196
rmt_rx_channel_config_t::invert_in (C++ member), 1196
rmt_rx_channel_config_t::io_loop_back
(C++ member), 1196
rmt_rx_channel_config_t::mem_block_symbols
(C++ member), 1196
rmt_rx_channel_config_t::resolution_hz
(C++ member), 1196
rmt_rx_channel_config_t::with_dma
(C++ member), 1196
rmt_tx_done_callback_t (C++ type), 1201
rmt_tx_done_event_data_t (C++ struct), 1201
rmt_tx_done_event_data_t::num_symbols
(C++ member), 1201
rmt_tx_done_event_data_t::received_symbols
(C++ member), 1201
rmt_tx_event_callbacks_t (C++ struct), 1195
rmt_tx_event_callbacks_t::on_recv_done
(C++ member), 1196
rmt_rx_register_event_callbacks (C++
function), 1195
rmt_symbol_word_t (C++ union), 1202
rmt_symbol_word_t::duration0 (C++ mem-
ber), 1202
rmt_symbol_word_t::duration1 (C++ mem-
ber), 1202
rmt_symbol_word_t::level0 (C++ member),
1202
rmt_symbol_word_t::level1 (C++ member),
1202
rmt_symbol_word_t::val (C++ member), 1202
rmt_symbol_word_t::[anonymous] (C++
member), 1202
rmt_sync_manager_config_t (C++
struct), 1194
rmt_sync_manager_config_t::array_size
(C++ member), 1194
rmt_sync_manager_config_t::tx_channel_config
(C++ member), 1194
rmt_sync_manager_handle_t (C++ type), 1201
rmt_sync_reset (C++ function), 1192
rmt_transmit (C++ function), 1191
rmt_transmit_config_t (C++ struct), 1194
rmt_transmit_config_t::committed_level
(C++ member), 1194
rmt_transmit_config_t::duration
(C++ member), 1194
rmt_transmit_config_t::flags (C++
member), 1193
rmt_tx_channel_config_t (C++ struct), 1193
rmt_tx_channel_config_t::clk_src
(C++ member), 1193
rmt_tx_channel_config_t::flags
(C++ member), 1193
rmt_tx_channel_config_t::gpio_num
(C++ member), 1193
rmt_tx_channel_config_t::invert_out
(C++ member), 1193
rmt_tx_channel_config_t::io_loop_back
(C++ member), 1193
rmt_tx_channel_config_t::mem_block_symbols
(C++ member), 1193
rmt_tx_channel_config_t::resolution_hz
(C++ member), 1193
rmt_tx_channel_config_t::trans_queue_depth
(C++ member), 1193
rmt_tx_done_callback_t (C++ type), 1201
rmt_tx_done_event_data_t (C++ struct), 1200
rmt_tx_done_event_data_t::num_symbols
(C++ member), 1201
rmt_tx_event_callbacks_t (C++ struct), 1193
rmt_tx_event_callbacks_t::on_trans_done
(C++ member), 1193
rmt_tx_register_event_callbacks (C++
function), 1191
rmt_tx_wait_all_done (C++ function), 1191
rtc_gpio_deinit (C++ function), 1017
rtc_gpio_force_hold_dis_all (C++ func-
tion), 1019
rtc_gpio_force_hold_en_all (C++ function),
1019
rtc_gpio_get_drive_capability (C++ func-
tion), 1018
rtc_gpio_get_level (C++ function), 1017
rtc_gpio_hold_dis (C++ function), 1019
rtc_gpio_hold_en (C++ function), 1018
rtc_gpio_init (C++ function), 1017
RTC_GPIO_IS_VALID_GPIO (C macro),
1020
rtc_gpio_is_valid_gpio (C++ function),
1016
rtc_gpio_isolate (C++ function), 1019
rtc_gpio_mode_t (C++ enum), 1020
arraygpio_mode_t::RTC_GPIO_MODE_DISABLED
(C++ enumerator), 1020
rtc_gpio_mode_t::RTC_GPIO_MODE_INPUT_ONLY
(C++ enumerator), 1020
rtc_gpio_mode_t::RTC_GPIO_MODE_INPUT_OUTPUT
(C++ enumerator), 1020
rtc_gpio_mode_t::RTC_GPIO_MODE_INPUT_OUTPUT_OD
(C++ enumerator), 1020
rtc_gpio_mode_t::RTC_GPIO_MODE_OUTPUT_OD
(C++ enumerator), 1020
rtc_gpio_mode_t::RTC_GPIO_MODE_OUTPUT_ONLY
(C++ enumerator), 1020
rtc_gpio_pulldown_dis (C++ function), 1018
rtc_gpio_pulldown_en (C++ function), 1018
rtc_gpio_pullup_dis (C++ function), 1018
rtc_gpio_pullup_en (C++ function), 1017
rtc_gpio_set_direction (C++ function),
1017
rtc_gpio_set_direction_in_sleep (C++
function), 1017
rtc_gpio_set_capabilty (C++ function),
rtc_gpio_wakeup_enable (C++ function), 1019
rtc_io_number_get (C++ function), 1016
RTC_SLOW_MEM (C macro), 2098

S

sdio_event_cb_t (C++ type), 1227
sdio_slave_buf_handle_t (C++ type), 1227
sdio_slave_clear_host_int (C++ function), 1225
sdio_slave_config_t (C++ struct), 1226
sdio_slave_config_t::event_cb (C++ member), 1226
sdio_slave_config_t::flags (C++ member), 1226
sdio_slave_config_t::recv_buffer_size (C++ member), 1226
sdio_slave_config_t::send_queue_size (C++ member), 1226
sdio_slave_config_t::sending_mode (C++ member), 1226
sdio_slave_config_t::timing (C++ member), 1226
sdio_slave_deinit (C++ function), 1222
SDIO_SLAVE_FLAG_DAT2_DISABLED (C macro), 1226
SDIO_SLAVE_FLAG_DEFAULT_SPEED (C macro), 1227
SDIO_SLAVE_FLAG_HOST_INTR_DISABLED (C macro), 1227
SDIO_SLAVE_FLAG_INTERNAL_PULLUP (C macro), 1226

sdio_slave_get_host_intena (C++ function), 1225
sdio_slave_hostint_t (C++ enum), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_SEND_NEW_PACKET (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_SEND_PACKET (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_SEND_STREAM (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_SEND_STREAM_NSAMPLE (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_PSEND_NSAMPLE (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_PSEND_PSAMPLE (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_NSEND_NSAMPLE (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_NSEND_PSAMPLE (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_PSEND_PSEND (C++ enumerator), 1221
sdio_slave_hostint_t::SDIO_SLAVE_HOSTINT_PSEND_PSEND_NSAMPLE (C++ enumerator), 1221

sdio_slave_recv_get_buf (C++ function), 1224
sdio_slave_recv_load_buf (C++ function), 1223
SDIO_SLAVE_RECV_MAX_BUFFER (C macro), 1226
sdio_slave_recv_packet (C++ function), 1223
sdio_slave_recv_register_buf (C++ function), 1223
sdio_slave_recv_unregister_buf (C++ function), 1223
sdio_slave_reset (C++ function), 1223
sdio_slave_send_get_finished (C++ function), 1224
sdio_slave_send_host_int (C++ function), 1225
sdio_slave_send_queue (C++ function), 1224
sdio_slave_send_mode (C++ enum), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_PACKET (C macro), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_PACKET_NSAMPLE (C macro), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_STREAM (C macro), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_STREAM_NSAMPLE (C macro), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_PSEND_NSAMPLE (C macro), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_PSEND_PSEND (C macro), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_PSEND_PSEND_NSAMPLE (C macro), 1222
sdio_slave_send_mode_t::SDIO_SLAVE_SEND_PSEND_PSEND_PSEND (C macro), 1222

sdio_slave_transmit (C++ function), 1225
sdio_slave_wait_int (C++ function), 1225
sdio_slave_write_reg (C++ function), 1225
sdm_channel_disable (C++ function), 1230
sdm_channel_enable (C++ function), 1230
sdm_channel_handle_t (C++ type), 1232
sdm_channel_set_duty (C++ function), 1231
sdm_clock_source_t (C++ type), 1232
sdm_config_t (C++ struct), 1231
sdm_config_t::clk_src (C++ member), 1231
sdm_config_t::flags (C++ member), 1232
sdm_config_t::gpio_num (C++ member), 1231
sdm_config_t::invert_out (C++ member), 1232
sdm_config_t::io_loop_back (C++ member), 1232
sdm_config_t::sample_rate_hz (C++ member), 1232
sdmmc_can_discard (C++ function), 1680
sdmmc_can_trim (C++ function), 1680
sdmmc_card_init (C++ function), 1678
sdmmc_card_print_info (C++ function), 1679
sdmmc_card_t (C++ struct), 1688
sdmmc_card_t::cid (C++ member), 1688
sdmmc_card_t::csd (C++ member), 1688
sdmmc_card_t::ext_csd (C++ member), 1688
sdmmc_card_t::host (C++ member), 1688
sdmmc_card_t::is_ddr (C++ member), 1689
sdmmc_card_t::is mmc (C++ member), 1689
sdmmc_card_t::is_sdio (C++ member), 1689
sdmmc_card_t::log_bus_width (C++ member), 1689
sdmmc_card_t::max_freq_khz (C++ member), 1688
sdmmc_card_t::num_io_functions (C++ member), 1689
sdmmc_card_t::ocr (C++ member), 1688
sdmmc_card_t::raw_cid (C++ member), 1688
sdmmc_card_t::real_freq_khz (C++ member), 1688
sdmmc_card_t::reserved (C++ member), 1689
sdmmc_card_t::scr (C++ member), 1688
sdmmc_card_t::ssr (C++ member), 1688
sdmmc_cid_t (C++ struct), 1684
sdmmc_cid_t::date (C++ member), 1684
sdmmc_cid_t::mfg_id (C++ member), 1684
sdmmc_cid_t::name (C++ member), 1684
sdmmc_cid_t::oem_id (C++ member), 1684
sdmmc_cid_t::revision (C++ member), 1684
sdmmc_cid_t::serial (C++ member), 1684
sdmmc_command_t (C++ struct), 1686
sdmmc_command_t::arg (C++ member), 1686
sdmmc_command_t::blklen (C++ member), 1686
sdmmc_command_t::datalen (C++ member), 1686
sdmmc_command_t::data (C++ member), 1686
sdmmc_command_t::error (C++ member), 1686
sdmmc_command_t::flags (C++ member), 1686
sdmmc_command_t::opcode (C++ member), 1686
sdmmc_command_t::response (C++ member), 1686
sdmmc_command_t::timeout_ms (C++ member), 1687
sdmmc_csd_t (C++ struct), 1683
sdmmc_csd_t::capacity (C++ member), 1684
sdmmc_csd_t::card_command_class (C++ member), 1684
sdmmc_csd_t::csd_ver (C++ member), 1683
sdmmc_csd_t::mmc_ver (C++ member), 1684
sdmmc_csd_t::read_block_len (C++ member), 1684
sdmmc_csd_t::sector_size (C++ member), 1684
sdmmc_csd_t::tr_speed (C++ member), 1684
sdmmc_erase_arg_t (C++ enum), 1690
sdmmc_erase_arg_t::SDMMC_DISCARD_ARG (C++ enumerator), 1690
sdmmc_erase_arg_t::SDMMC_ERASE_ARG (C++ enumerator), 1690
sdmmc_erase_sectors (C++ function), 1679
sdmmc_erase_sectors (C++ struct), 1685
sdmmc_erase_sectors::erase_mem_state (C++ member), 1686
sdmmc_erase_sectors::power_class (C++ member), 1686
sdmmc_erase_sectors::rev (C++ member), 1686
sdmmc_erase_sectors::sec_feature (C++ member), 1686
SDMMC_FREQ_26M (C macro), 1690
SDMMC_FREQ_52M (C macro), 1690
SDMMC_FREQ_DEFAULT (C macro), 1689
SDMMC_FREQ_HIGHSPEED (C macro), 1689
SDMMC_FREQ_PROBING (C macro), 1689
sdmmc_full_erase (C++ function), 1680
sdmmc_get_status (C++ function), 1679
SDMMC_HOST_DEFAULT (C macro), 1211
sdmmc_host_deinit (C++ function), 1210
sdmmc_host_do_transaction (C++ function), 1209
SDMMC_HOST_FLAG_1BIT (C macro), 1689
SDMMC_HOST_FLAG_4BIT (C macro), 1689
SDMMC_HOST_FLAG_8BIT (C macro), 1689
SDMMC_HOST_FLAG_DDR (C macro), 1689
SDMMC_HOST_FLAG_DEINIT_ARG (C macro), 1689
SDMMC_HOST_FLAG_SPI (C macro), 1689
sdmmc_host_get_real_freq (C++ function), 1210
sdmmc_host_get_slot_width (C++ function), 1209
sdmmc_host_init (C++ function), 1208
sdmmc_host_init_slot (C++ function), 1208
sdmmc_host_io_int_enable (C++ function), 1210
sdmmc_host_io_int_wait (C++ function), 1210
sdmmc_host_set_bus_ddr_mode (C++ function), 1209
sdmmc_host_set_bus_width (C++ function), 1208
sdmmc_host_set_card_clk (C++ function), 1209
SDMMC_HOST_SLOT_0 (C macro), 1211
SDMMC_HOST_SLOT_1 (C macro), 1211
sdmmc_host_t (C++ struct), 1687
sdmmc_host_t::command_timeout_ms (C++ member), 1688
sdmmc_host_t::command_timeout_ms (C++ member), 1688
sdmmc_host_t::deinit (C++ member), 1687
sdmmc_host_t::deinit_p (C++ member), 1687
sdmmc_host_t::do_transaction (C++ member), 1687
sdmmc_host_t::flags (C++ member), 1687
sdmmc_host_t::get_bus_width (C++ member), 1687

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>soc_periph_sdm_clk_src_t::SDM_CLK_SRC_APB</td>
<td>Soc_periph_sdm_clk_src_t enum, 2056</td>
</tr>
<tr>
<td>soc_periph_dac_digi_clk_src_t::DAC_DIGI_CLK_SRC_PLLD2</td>
<td>Soc_periph_dac_digi_clk_src_t enum, 987</td>
</tr>
<tr>
<td>soc_periph_dac_digi_clk_src_t::DAC_DIGI_CLK_SRC_APLL</td>
<td>Soc_periph_dac_digi_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_dac_digi_clk_src_t::DAC_DIGI_CLK_SRC_DEFAULT</td>
<td>Soc_periph_dac_digi_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_dac_cosine_clk_src_t::DAC_COSINE_CLK_SRC_RTC_SLOW</td>
<td>Soc_periph_dac_cosine_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_dac_cosine_clk_src_t::DAC_COSINE_CLK_SRC_DEFAULT</td>
<td>Soc_periph_dac_cosine_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_dac_cosine_clk_src_t::DAC_COSINE_CLK_SRC_RTC_FAST</td>
<td>Soc_periph_dac_cosine_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_dac_cosine_clk_src_t::DAC_COSINE_CLK_SRC_DEFAULT</td>
<td>Soc_periph_dac_cosine_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_lcd_clk_src_t::LCD_CLK_SRC_PLL160M</td>
<td>Soc_periph_lcd_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_lcd_clk_src_t::LCD_CLK_SRC_APLL</td>
<td>Soc_periph_lcd_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_lcd_clk_src_t::LCD_CLK_SRC_DEFAULT</td>
<td>Soc_periph_lcd_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_i2s_clk_src_t::I2S_CLK_SRC_PLL_160M</td>
<td>Soc_periph_i2s_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_i2s_clk_src_t::I2S_CLK_SRC_APLL</td>
<td>Soc_periph_i2s_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_i2s_clk_src_t::I2S_CLK_SRC_DEFAULT</td>
<td>Soc_periph_i2s_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_i2c_clk_src_t::I2C_CLK_SRC_APB</td>
<td>Soc_periph_i2c_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_DEFAULT</td>
<td>Soc_periph_gptimer_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_APB</td>
<td>Soc_periph_gptimer_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_gptimer_clk_src_t::GPTIMER_CLK_SRC_DEFAULT</td>
<td>Soc_periph_gptimer_clk_src_t enum, 984</td>
</tr>
<tr>
<td>soc_periph_sdmc_clk_src_t::SDM_CLK_SRC_APB</td>
<td>Soc_periph_sdmc_clk_src_t enum, 984</td>
</tr>
</tbody>
</table>
Espressif Systems

(C++ enumerator), 987

soc_periph_sdm_clk_src_t::SDM_CLK_DOWNSCALE
(C++ enumerator), 987

soc_periph_systimer_clk_src_t::SYSTIMER_CLK_DOWNSCALE
(C++ enum), 984

soc_periph_twai_clk_src_t::TWAI_CLK_DOWNSCALE
(C++ enum), 984

soc_periph_tg_clk_src_legacy_t::TIMER_SRC_CLK_DOWNSCALE
(C++ enum), 984

soc_periph_systimer_clk_src_t::SYSTIMER_CLK_UPSCALE
(C++ enum), 984

soc_periph_sdm_clk_src_t::SDM_CLK_UPSCALE
(C++ enum), 984

soc_rtc_slow_clk_src_t::SOC_RTC_SLOW_CLK_SRC_XTAL32K
(C++ macro), 987

soc_root_clk_t::SOC_ROOT_CLK_INT_RC_SLOW
(C++ macro), 982

soc_root_clk_t::SOC_ROOT_CLK_INT_RC_FAST
(C++ macro), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL
(C++ macro), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL32K
(C++ macro), 982

soc_periph_uart_clk_src_legacy_t::UART_SCLK_DEFAULT
(C macro), 2058

soc_periph_uart_clk_src_legacy_t::UART_SCLK_APB
(C macro), 2058

soc_periph_twai_clk_src_t::TWAI_CLK_SRC_DEFAULT
(C macro), 2058

soc_periph_twai_clk_src_t::TWAI_CLK_SRC_APB
(C macro), 2058

soc_periph_tg_clk_src_legacy_t::TIMER_SRC_CLK_DEFAULT
(C macro), 2058

soc_periph_tg_clk_src_legacy_t::TIMER_SRC_CLK_APB
(C macro), 2058

SOC_PHY_DIG_REGS_MEM_SIZE (C macro), 2058

SOC_PM_SUPPORT_EXT_WAKEUP (C macro), 2058

SOC_PM_SUPPORT_RTC_FAST_MEM_PD (C macro), 2058

SOC_PM_SUPPORT_RTC_PERIPH_PD (C macro), 2058

SOC_PM_SUPPORT_RTC_SLOW_MEM_PD (C macro), 2058

SOC_PM_SUPPORT_TOUCH_SENSOR_WAKEUP (C macro), 2058

SOC_REG_TO_ULP_PERIPH_SEL (C++ function), 2081

SOC_RMT_CHANNEL_CLK_INDEPENDENT (C macro), 2055

SOC_RMT_CHANNELS_PER_GROUP (C macro), 2055

SOC_RMT_CLKS (C macro), 981

SOC_RMT_GROUPS (C macro), 2055

SOC_RMT_MEM_WORDS_PER_CHANNEL (C macro), 2055

SOC_RMT_RX_CANDIDATES_PER_GROUP (C macro), 2055

SOC_RMT_SUPPORT_APB (C macro), 2055

SOC_RMT_SUPPORT_REF_TICK (C macro), 2055

SOC_RMT_SUPPORTED (C macro), 2051

SOC_RST_CANDIDATES_PER_GROUP (C macro), 2055

soc_root_clk_t (C++ enum), 981

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL
(C++ enum), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL32K
(C++ enum), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL16
(C++ enum), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL12
(C++ enum), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL8
(C++ enum), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL4
(C++ enum), 982

soc_root_clk_t::SOC_ROOT_CLK_EXT_XTAL2
(C++ enum), 982
spi_device_polling_start

spi_device_interface_config_t::spics_io_num
spi_device_interface_config_t::queue_size
spi_device_interface_config_t::pre_cb
spi_device_interface_config_t::post_cb
spi_device_interface_config_t::mode
spi_device_interface_config_t::input_delay_ns
spi_device_interface_config_t::flags
spi_device_interface_config_t::cs_ena_posttrans
spi_device_interface_config_t::command_bits
spi_device_interface_config_t::clock_speed_hz
spi_device_interface_config_t::address_bits

SPI_DEVICE_DDRCLK
SPI_DEVICE_BIT_LSBFIRST

spi_device_acquire_bus

spi_common_dma_t::SPI_DMA_DISABLED
spi_common_dma_t::SPI_DMA_CH2
spi_common_dma_t::SPI_DMA_CH1

spi_device_polling_transmit

SPI_DEVICE_NO_DUMMY (C macro), 1256
SPI_DEVICE_NO_RETURN_RESULT (C macro), 1256

spi_device_polling_end
spi_device_polling_start

SPI_DEVICE_POSITIVE_CS (C macro), 1256
spi_device_queue_trans
spi_device_release_bus
SPI_DEVICE_RXBIT_LSBFIRST (C macro), 1255
spi_device_transmit
SPI_DEVICE_TXBIT_LSBFIRST (C macro), 1255
spi_dma_chan_t (C++ type), 1248
spi_event_t (C++ enum), 1243
spi_event_t::SPI_EVT_BUF_RX (C++ enumerator), 1243
spi_event_t::SPI_EVT_BUF_TX (C++ enumerator), 1243
spi_event_t::SPI_EVT_CMD9 (C++ enumerator), 1243
spi_event_t::SPI_EVT_CMDA (C++ enumerator), 1244
spi_event_t::SPI_EVT_RECV (C++ enumerator), 1244
spi_event_t::SPI_EVT_RECV_DMA_READY (C++ enumerator), 1243
spi_event_t::SPI_EVT_SEND (C++ enumerator), 1243
spi_event_t::SPI_EVT_SEND_DMA_READY (C++ enumerator), 1243

spi_flash_host_driver_s::configure_host_io_mode
spi_flash_host_driver_s::dev_config
spi_flash_host_driver_s::erase_block
spi_flash_host_driver_s::erase_chip
spi_flash_host_driver_s::erase_chip
spi_flash_host_driver_s::dev_config
spi_flash_host_driver_s::configure_host_io_mode
spi_flash_host_driver_s::erase_block
spi_flash_host_driver_s::erase_chip
touch_pad_t::TOUCH_PAD_NUM9 (C++ enumerator), 1279
TOUCH_PAD_THRESHOLD_MAX (C macro), 1278
TOUCH_PAD_TIE_OPT_DEFAULT (C macro), 1277
touch_tie_opt_t (C++ enum), 1281
twai_clock_source_t (C++ type), 1294
twai_driver_install (C++ function), 1295
twai_driver_uninstall (C++ function), 1295
twai_clear_transmit_queue (C++ function), 1297
twai_initiate_recovery (C++ function), 1297
twai_clear_receive_queue (C++ function), 1297
TSKMPU_REGION_VIEW (C macro), 1290
TSKKERNEL_VERSION_NUMBER (C macro), 1290
twkClearReceiveQueue (C++ function), 1290
twai_clear_transmit_queue (C++ function), 1297
TWAI_FRAME_STD_ID_LEN_BYTES (C macro), 1294
TWAI_FRAME_MAX_ID_LEN_BYTES (C macro), 1294
twai_LPMS_config_t::max_transfer_time (C++ member), 1298
twai_filter_config_t::acceptance_code (C++ member), 1293
twai_filter_config_t::acceptance_mask (C++ member), 1293
twai_filter_config_t::single_filter (C++ member), 1293
TWAI_FRAME_MAX_DLC (C macro), 1294
TWAI_FRAME_EXTD_ID_LEN_BYTES (C macro), 1294
TWAI_FRAME_STD_ID_LEN_BYTES (C macro), 1294
twai_general_config_t::bus_off_io (C++ member), 1299
twai_general_config_t::alerts_enabled (C++ member), 1299
twai_general_config_t::clkout_divider (C++ member), 1299
twai_general_config_t::clkout_io (C++ member), 1298
twai_general_config_t::intr_flags (C++ member), 1299
twai_general_config_t::mode (C++ member), 1298
twai_general_config_t::rx_queue_length (C++ member), 1298
twai_general_config_t::tx_queue_length (C++ member), 1298
twai_general_config_t::tx_queue_length (C++ member), 1298
transaction_cb_t (C++ type), 1257
twai_get_status_info (C++ function), 1297
twai_initiate_recovery (C++ function), 1297
TWAI_TO_UNUSED (C macro), 1300
twai_message_t (C++ struct), 1298
twai_message_t::rtr (C++ enum), 1298
twai_message_t::reserved (C++ enum), 1298
twai_message_t::identifier (C++ enum), 1298
twai_message_t::data_length_code (C++ enum), 1298
twai_message_t::data (C++ enum), 1298
twai_message_t::dlc_non_comp (C++ enum), 1298
twai_message_t::extd (C++ enum), 1292
twai_message_t::flags (C++ enum), 1292
twai_message_t::data_length_code (C++ enum), 1292
twai_message_t::data_length_code (C++ enum), 1292
twai_message_t::data_length_code (C++ enum), 1292
twai_message_t::reserved (C++ enum), 1292
twai_message_t::rtr (C++ enum), 1292
twai_message_t::self (C++ enum), 1292
twai_message_t::ss (C++ member), 1292

twai_mode_t (C++ enum), 1294
twai_mode_t::TWAI_MODE_LISTEN_ONLY (C++ enum), 1294
twai_mode_t::TWAI_MODE_NO_ACK (C++ enum), 1294
twai_mode_t::TWAI_MODE_NORMAL (C++ enum), 1294
twai_read_alerts (C++ function), 1296
twai_receive (C++ function), 1296

twai_reconfigure_alerts (C++ function), 1297

twai_start (C++ function), 1295
twai_state_t (C++ enum), 1300
twai_state_t::TWAI_STATE_BUS_OFF (C++ enum), 1300
twai_state_t::TWAI_STATE_RECOVERING (C++ enum), 1300
twai_state_t::TWAI_STATE_RUNNING (C++ enum), 1300
twai_state_t::TWAI_STATE_STOPPED (C++ enum), 1300
twai_status_info_t (C++ struct), 1299
twai_status_info_t::arb_lost_count (C++ member), 1299
twai_status_info_t::bus_error_count (C++ member), 1299
twai_status_info_t::msgs_to_rx (C++ member), 1299
twai_status_info_t::msgs_to_tx (C++ member), 1299
twai_status_info_t::rx_error_counter (C++ member), 1299
twai_status_info_t::rx_missed_count (C++ member), 1299
twai_status_info_t::rx_overrun_count (C++ member), 1299
twai_status_info_t::state (C++ member), 1299
twai_status_info_t::tx_error_counter (C++ member), 1299
twai_status_info_t::tx_failed_count (C++ member), 1299
twai_timing_config_t (C++ struct), 1292

twai_timing_config_t::brp (C++ member), 1293
twai_timing_config_t::clk_src (C++ member), 1293
twai_timing_config_t::quanta_resolution (C++ member), 1293
twai_timing_config_t::sjw (C++ member), 1293
twai_timing_config_t::tseg_1 (C++ member), 1293
twai_timing_config_t::tseg_2 (C++ member), 1293
twai_transmit (C++ function), 1296

twai_message_t::ss (C++ member), 1292

twai_mode_t (C++ enum), 1294

twai_mode_t::TWAI_MODE_LISTEN_ONLY (C++ enum), 1294

twai_mode_t::TWAI_MODE_NO_ACK (C++ enum), 1294

twai_mode_t::TWAI_MODE_NORMAL (C++ enum), 1294

twai_read_alerts (C++ function), 1296
twai_receive (C++ function), 1296

twai_reconfigure_alerts (C++ function), 1297

twai_start (C++ function), 1295
twai_state_t (C++ enum), 1300
twai_state_t::TWAI_STATE_BUS_OFF (C++ enum), 1300
twai_state_t::TWAI_STATE_RECOVERING (C++ enum), 1300
twai_state_t::TWAI_STATE_RUNNING (C++ enum), 1300
twai_state_t::TWAI_STATE_STOPPED (C++ enum), 1300

twai_status_info_t (C++ struct), 1299
twai_status_info_t::arb_lost_count (C++ member), 1299
twai_status_info_t::bus_error_count (C++ member), 1299
twai_status_info_t::msgs_to_rx (C++ member), 1299
twai_status_info_t::msgs_to_tx (C++ member), 1299
twai_status_info_t::rx_error_counter (C++ member), 1299
twai_status_info_t::rx_missed_count (C++ member), 1299
twai_status_info_t::rx_overrun_count (C++ member), 1299
twai_status_info_t::state (C++ member), 1299
twai_status_info_t::tx_error_counter (C++ member), 1299
twai_status_info_t::tx_failed_count (C++ member), 1299

UART

twai_timing_config_t::tseg_1 (C++ member), 1293
uart_event_type_t::UART_BUFFER_FULL (C++ enumerator, 1319)
uart_event_type_t::UART_DATA (C++ enumerator, 1319)
uart_event_type_t::UART_DATA_BREAK (C++ enumerator, 1319)
uart_event_type_t::UART_EVENT_MAX (C++ enumerator, 1319)
uart_event_type_t::UART_FIFO_OVF (C++ enumerator, 1319)
uart_event_type_t::UART_FRAME_OVF (C++ enumerator, 1319)
uart_event_type_t::UART_PARITY_ERR (C++ enumerator, 1319)
uart_event_type_t::UART_PATTERN_DET (C++ enumerator, 1319)
UART_GPIO9_DIRECT_CHANNEL (C macro, 1318)
uart_flush (C++ function, 1312)
uart_flush_input (C++ function, 1313)
uart_get_baudrate (C++ function, 1308)
uart_get_buffered_data_len (C++ function, 1313)
uart_get_collision_flag (C++ function, 1316)
uart_get_hw_flowctrl (C++ function, 1309)
uart_get_parity (C++ function, 1307)
uart_get_scik_freq (C++ function, 1308)
uart_get_stop_bits (C++ function, 1307)
uart_get_tx_buffer_free_size (C++ function, 1313)
uart_get_wakeup_threshold (C++ function, 1316)
UART_GPIO10_DIRECT_CHANNEL (C macro, 1324)
UART_GPIO11_DIRECT_CHANNEL (C macro, 1324)
UART_GPIO16_DIRECT_CHANNEL (C macro, 1325)
UART_GPIO17_DIRECT_CHANNEL (C macro, 1325)
UART_GPIO19_DIRECT_CHANNEL (C macro, 1324)
UART_GPIO122_DIRECT_CHANNEL (C macro, 1324)
UART_GPIO3_DIRECT_CHANNEL (C macro, 1324)
UART_GPIO6_DIRECT_CHANNEL (C macro, 1324)
UART_GPIO7_DIRECT_CHANNEL (C macro, 1325)
UART_GPIO8_DIRECT_CHANNEL (C macro, 1325)
UART_GPIO9_DIRECT_CHANNEL (C macro, 1324)
uart_hw_flowcontrol_t (C++ enum), 1322
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_CTS (macro, 1325)
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_CTS_R (macro, 1325)
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_DISABLE (C++ function, 1311)
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_DISABLE (C++ function, 1311)
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_DISABLE (C++ function, 1311)
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_DISABLE (C++ function, 1311)
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_DISABLE (C++ function, 1311)
uart_hw_flowcontrol_t::UART_HW_FLOWCTRL_DISABLE (C++ function, 1311)
uart_intr_config (C++ function), 1311
uart_intr_config_t (C++ struct), 1317
uart_intr_config_t::intr_enable_mask (C++ member), 1317
uart_intr_config_t::rx_timeout_thresh (C++ member), 1317
uart_intr_config_t::xffifo_full_thresh (C++ member), 1318
uart_intr_config_t::xffifo_empty_intr_thresh (C++ member), 1317
uart_is_driver_installed (C++ function), 1307
uart_isr_handle_t (C++ type, 1318)
uart_mode_t::UART_MODE_IRDA (C++ enumerator, 1321)
uart_mode_t::UART_MODE_RS485_APP_CTRL (C++ enumerator, 1321)
uart_mode_t::UART_MODE_RS485_COLLISION_DETECT (C++ enumerator, 1321)
uart_mode_t::UART_MODE_RS485_HALF_DUPLEX (C++ enumerator, 1321)
uart_mode_t::UART_MODE_UART (C++ enumerator, 1321)
UART_NUM_0 (C macro, 1318)
UART_NUM_0_CTS_DIRECT_GPIO_NUM (C macro, 1324)
UART_NUM_0_RTS_DIRECT_GPIO_NUM (C macro, 1324)
UART_NUM_0_TXD_DIRECT_GPIO_NUM (C macro, 1324)
UART_NUM_1 (C macro, 1318)
UART_NUM_1_CTS_DIRECT_GPIO_NUM (C macro, 1324)
UART_NUM_1_RTS_DIRECT_GPIO_NUM (C macro, 1325)
UART_NUM_1_TXD_DIRECT_GPIO_NUM (C macro, 1324)
UART_NUM_2 (C macro, 1318)
UART_NUM_2_CTS_DIRECT_GPIO_NUM (C macro, 1325)
UART_NUM_2_RTS_DIRECT_GPIO_NUM (C macro, 1325)
uart_parity_t::UART_PARITY_EVEN (C++ enumerator, 1322)
uart_parity_t::UART_PARITY_ODD (C++ enumerator, 1322)
ulp_insn::cmp (C++ member), 2083
ulp_insn::cycle_sel (C++ member), 2085
ulp_insn::cycles (C++ member), 2081
ulp_insn::data (C++ member), 2084
ulp_insn::delay (C++ member), 2081
ulp_insn::dreg (C++ member), 2081
ulp_insn::end (C++ member), 2085
ulp_insn::halt (C++ member), 2082
ulp_insn::high (C++ member), 2084
ulp_insn::high_bits (C++ member), 2084
ulp_insn::i2c (C++ member), 2085
ulp_insn::i2c_addr (C++ member), 2084
ulp_insn::i2c_sel (C++ member), 2084
ulp_insn::imm (C++ member), 2083
ulp_insn::instruction (C++ member), 2085
ulp_insn::label (C++ member), 2085
ulp_insn::ld (C++ member), 2082
ulp_insn::low (C++ member), 2084
ulp_insn::low_bits (C++ member), 2084
ulp_insn::macro (C++ member), 2085
ulp_insn::opcode (C++ member), 2084
ulp_insn::periph_sel (C++ member), 2083
ulp_insn::rd_reg (C++ member), 2084
ulp_insn::reg (C++ member), 2083
ulp_insn::reserved (C++ member), 2084
ulp_insn::rw (C++ member), 2084
ulp_insn::sar_sel (C++ member), 2084
ulp_insn::sel (C++ member), 2083
ulp_insn::sign (C++ member), 2083
ulp_insn::sleep (C++ member), 2085
ulp_insn::sreg (C++ member), 2082
ulp_insn::st (C++ member), 2082
ulp_insn::sub_opcode (C++ member), 2082
ulp_insn::treg (C++ member), 2083
ulp_insn::tsens (C++ member), 2084
ulp_insn::type (C++ member), 2083
ulp_insn::unused (C++ member), 2081
ulp_insn::unused1 (C++ member), 2082
ulp_insn::unused2 (C++ member), 2082
ulp_insn::wait_delay (C++ member), 2084
ulp_insn::wakeup (C++ member), 2085
ulp_insn::wr_reg (C++ member), 2084
ulp_insn::t (C++ type), 2097
ulp_load_binary (C++ function), 2096
ulp_process_macros_and_load (C++ function), 2096
ulp_run (C++ function), 2096
ulp_set_wakeup_period (C++ function), 2097
ulp_timer_resume (C++ function), 2098
ulp_timer_stop (C++ function), 2097
ulTaskGenericNotifyTake (C++ function), 1843
ulTaskGenericNotifyValueClear (C++ function), 1844
ulTaskGetIdleRunTimeCounter (C++ function), 1838
ulTaskNotifyTake (C macro), 1849
ulTaskNotifyTakeIndexed (C macro), 1849
ulTaskNotifyValueClear (C macro), 1849
ulTaskNotifyValueClearIndexed (C macro), 1849
uxQueueMessagesWaiting (C++ function), 1854
uxQueueMessagesWaitingFromISR (C++ function), 1857
uxQueueSpacesAvailable (C++ function), 1854
uxSemaphoreGetCount (C macro), 1883
uxTaskGetNumberOfTasks (C++ function), 1833
uxTaskGetStackHighWaterMark (C++ function), 1833
uxTaskGetStackHighWaterMark2 (C++ function), 1833
uxTaskGetSystemState (C++ function), 1835
uxTaskPriorityGet (C++ function), 1826
uxTaskPriorityGetFromISR (C++ function), 1827
uxTimerGetReloadMode (C++ function), 1891

V
vApplicationGetIdleTaskMemory (C++ function), 1835
vApplicationGetTimerTaskMemory (C++ function), 1891
vendor_ie_data_t (C++ struct), 860
vendor_ie_data_t::element_id (C++ member), 861
vendor_ie_data_t::length (C++ member), 861
vendor_ie_data_t::payload (C++ member), 861
vendor_ie_data_t::vendor_oui (C++ member), 861
vendor_ie_data_t::vendor_oui_type (C++ member), 861
vEventGroupDelete (C++ function), 1908
vMessageBufferDelete (C macro), 1925
vprintf_like_t (C++ type), 1997
vQueueAddToRegistry (C++ function), 1857
vQueueDelete (C++ function), 1854
vQueueUnregisterQueue (C++ function), 1857
vRingbufferDelete (C++ function), 1942
vRingbufferGetInfo (C++ function), 1943
vRingbufferReturnItem (C++ function), 1942
vRingbufferReturnItemFromISR (C++ function), 1942
vSemaphoreCreateBinary (C macro), 1870
vSemaphoreDelete (C macro), 1883
vStreamBufferDelete (C++ function), 1915
vTaskAllocateMPURegions (C++ function), 1822
vTaskDelay (C++ function), 1824
vTaskDelayUntil (C macro), 1847
vTaskDelete (C++ function), 1824
vTaskEndScheduler (C++ function), 1830
<table>
<thead>
<tr>
<th>WiFi error reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIFI_REASON_INVALID_FT_ACTION</td>
</tr>
<tr>
<td>WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT</td>
</tr>
<tr>
<td>WIFI_REASON_INVALID_FTE</td>
</tr>
<tr>
<td>WIFI_REASON_802_1X_AUTH_FAIL</td>
</tr>
<tr>
<td>WIFI_REASON_INVALID_MDE</td>
</tr>
<tr>
<td>WIFI_REASON_AKMP_INVALID</td>
</tr>
<tr>
<td>WIFI_REASON_ALTERATIVE_CHANNEL_OCCUPIED</td>
</tr>
<tr>
<td>WIFI_REASON_INVALID_RSN_IE_CAP</td>
</tr>
<tr>
<td>WIFI_REASON_TDLS_UNSPECIFIED</td>
</tr>
<tr>
<td>WIFI_REASON_TDLS_PEER_UNREACHABLE</td>
</tr>
<tr>
<td>WIFI_REASON_STA_LEAVING</td>
</tr>
<tr>
<td>WIFI_REASON_SSP_REQUESTED_DISASSOC</td>
</tr>
<tr>
<td>WIFI_REASON_SERVICE_CHANGE_PERLINKS_TS</td>
</tr>
<tr>
<td>WIFI_REASON_ROAMING</td>
</tr>
<tr>
<td>WIFI_REASON_BSS_TRANSITION_DISASSOC</td>
</tr>
<tr>
<td>WIFI_REASON книг</td>
</tr>
<tr>
<td>WIFI_REASON_INVALID_PMKID</td>
</tr>
<tr>
<td>WIFI_REASON_INVALID_MDE</td>
</tr>
<tr>
<td>WIFI_REASON_INVALID_FTE</td>
</tr>
</tbody>
</table>

C++ macro

wifi_err_reason_t::WIFI_REASON_UNSUPP_RSN_IE_VERSION
wifi_err_reason_t::WIFI_REASON_UNSPECIFIED
wifi_err_reason_t::WIFI_REASON_UNKNOWN_BA
wifi_err_reason_t::WIFI_REASON_TRANSMISSION_LINK_ESTABLISH_FAILED
wifi_err_reason_t::WIFI_REASON_TIMEOUT
wifi_err_reason_t::WIFI_REASON_TDLS_UNSPECIFIED
wifi_err_reason_t::WIFI_REASON_STA_LEAVING
wifi_err_reason_t::WIFI_REASON_SSP_REQUESTED_DISASSOC
wifi_err_reason_t::WIFI_REASON_SERVICE_CHANGE_PERLINKS_TS
wifi_err_reason_t::WIFI_REASON_ROAMING
wifi_err_reason_t::WIFI_REASON_BSS_TRANSITION_DISASSOC
wifi_err_reason_t::WIFI_REASON книга |
wifi_err_reason_t::WIFI_REASON_INVALID_PMKID |
wifi_err_reason_t::WIFI_REASON_INVALID_MDE |
wifi_err_reason_t::WIFI_REASON_INVALID_FTE |

C++ enumeration

C++ member

<table>
<thead>
<tr>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>wifi_event_action_tx_status_t::context</td>
</tr>
</tbody>
</table>

C++ struct

C++ enum

C++ enumerator

Release v5.1-dev-2066-g7869f4e151
wifi_event_action_tx_status_t::da (C++ member), 870
wifi_event_action_tx_status_t::ifix (C++ member), 870
wifi_event_action_tx_status_t::status (C++ member), 870
wifi_event_ap_probe_req_rx_t (C++ struct), 869
wifi_event_ap_probe_req_rx_t::mac (C++ member), 869
wifi_event_ap_probe_req_rx_t::rssi (C++ member), 869
wifi_event_ap_stadisconnected_t (C++ struct), 868
wifi_event_ap_stadisconnected_t::aid (C++ member), 868
wifi_event_ap_stadisconnected_t::is_mesh_child (C++ member), 868
wifi_event_ap_stadisconnected_t::mac (C++ member), 868
wifi_event_ap_stadisconnected_t (C++ struct), 868
wifi_event_ap_stadisconnected_t::aid (C++ member), 868
wifi_event_ap_stadisconnected_t::is_mesh_child (C++ member), 868
wifi_event_ap_stadisconnected_t::mac (C++ member), 868
wifi_event_bss_rssi_low_t (C++ struct), 869
wifi_event_roc_done_t::context (C++ member), 871
wifi_event_sta_authmode_change_t (C++ struct), 867
wifi_event_sta_authmode_change_t::context (C++ member), 867
wifi_event_sta_authmode_change_t::new_mode (C++ member), 867
wifi_event_sta_authmode_change_t::old_mode (C++ member), 867
wifi_event_sta_connected_t (C++ struct), 866
wifi_event_sta_connected_t::authmode (C++ member), 867
wifi_event_sta_connected_t::bssid (C++ member), 867
wifi_event_sta_connected_t::channel (C++ member), 867
wifi_event_sta_connected_t::ssid (C++ member), 866
wifi_event_sta_connected_t::ssid_len (C++ member), 866
wifi_event_sta_disconnected_t (C++ struct), 867
wifi_event_sta_disconnected_t::bssid (C++ member), 867
wifi_event_sta_disconnected_t::reason (C++ member), 867
wifi_event_sta_disconnected_t::rssi (C++ member), 867
wifi_event_sta_disconnected_t::ssid (C++ member), 867
wifi_event_sta_disconnected_t::ssid_len (C++ member), 867
wifi_event_sta_scan_done_t (C++ struct), 866
wifi_event_sta_scan_done_t::number (C++ member), 866
wifi_event_sta_scan_done_t::scan_id (C++ member), 866
wifi_event_sta_scan_done_t::status (C++ member), 866
wifi_event_sta_wps_er_pin_t (C++ struct), 867
wifi_event_sta_wps_er_pin_t::pin_code (C++ member), 868
wifi_event_sta_wps_er_success_t (C++ struct), 868
wifi_event_sta_wps_er_success_t::ap_cred (C++ member), 868
wifi_event_sta_wps_er_success_t::ap_cred_cnt (C++ member), 868
wifi_event_sta_wps_er_success_t::passphrase (C++ member), 868
wifi_event_sta_wps_er_success_t::ssid (C++ member), 868
wifi_event_sta_wps_fail_reason_t (C++ enum), 885
wifi_event_sta_wps_fail_reason_t::WPS_FAIL_REASON (C++ enumerator), 885
wifi_event_sta_wps_fail_reason_t::WPS_FAIL_REASON (C++ enumerator), 885
wifi_event_sta_wps_fail_reason_t::WPS_FAIL_REASON (C++ enumerator), 885
wifi_event_sta_wps_fail_reason_t::WPS_FAIL_REASON (C++ enumerator), 885
wifi_event_sta_wps_fail_reason_t::WPS_FAIL_REASON (C++ enumerator), 885
<table>
<thead>
<tr>
<th>wifi_init_config_t::sta_disconnected_p</th>
<th>wifi_phy_rate_t::WIFI_PHY_RATE_5M_S</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C++ member), 851</td>
<td>(C++ enumerator), 881</td>
</tr>
<tr>
<td>wifi_init_config_t::static_rx_buf_num</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_6M</td>
</tr>
<tr>
<td>(C++ member), 850</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_init_config_t::static_tx_buf_num</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_9M</td>
</tr>
<tr>
<td>(C++ member), 850</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_init_config_t::tx_buf_type</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_LORA_250K</td>
</tr>
<tr>
<td>(C++ member), 850</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>wifi_init_config_t::wifi_task_core_id</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_LORA_500K</td>
</tr>
<tr>
<td>(C++ member), 851</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>wifi_interface_t (C++ enum), 873</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MAX</td>
</tr>
<tr>
<td>wifi_interface_t::WIFI_IF_STA</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>(C++ enumerator), 873</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS0_LGI</td>
</tr>
<tr>
<td>wifi_mode_t (C++ enum), 873</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_mode_t::WIFI_MODE_AP</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS0_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 873</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_mode_t::WIFI_MODE_APSTA</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS1_LGI</td>
</tr>
<tr>
<td>(C++ enumerator), 873</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_mode_t::WIFI_MODE_MAX</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS1_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 873</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_mode_t::WIFI_MODE_NULL</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS2_LGI</td>
</tr>
<tr>
<td>(C++ enumerator), 873</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_mode_t::WIFI_MODE_STA</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS2_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 873</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_11M_L</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS3_LGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_11M_S</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS3_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_12M</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS4_LGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_18M</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS4_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 882</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_1M_L</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS5_LGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_1M_S</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS5_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_24M</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS6_LGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_2M_L</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS6_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_2M_S</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS7_LGI</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>(C++ enumerator), 882</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_36M</td>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_MCS7_SGI</td>
</tr>
<tr>
<td>(C++ enumerator), 882</td>
<td>(C++ enumerator), 883</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_48M</td>
<td>wifi_pkt_rx_ctrl_t (C++ struct), 861</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>wifi_pkt_rx_ctrl_t::<strong>pad0</strong> (C++ member), 861</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_54M</td>
<td>wifi_pkt_rx_ctrl_t::<strong>pad1</strong> (C++ member), 861</td>
</tr>
<tr>
<td>(C++ enumerator), 882</td>
<td>wifi_pkt_rx_ctrl_t::<strong>pad2</strong> (C++ member), 861</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_5M_L</td>
<td>wifi_pkt_rx_ctrl_t::<strong>pad3</strong> (C++ member), 862</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>wifi_pkt_rx_ctrl_t::<strong>pad4</strong> (C++ member), 862</td>
</tr>
<tr>
<td>wifi_phy_rate_t::WIFI_PHY_RATE_5M_S</td>
<td>wifi_pkt_rx_ctrl_t::<strong>pad5</strong> (C++ member), 862</td>
</tr>
<tr>
<td>(C++ enumerator), 881</td>
<td>wifi_pkt_rx_ctrl_t::<strong>pad6</strong> (C++ member), 862</td>
</tr>
</tbody>
</table>
WIFI_PROMIS_CTRL_FILTER_MASK_RTS (C macro), 872
WIFI_PROMIS_CTRL_FILTER_MASK_WARPPER (C macro), 872
WIFI_PROMIS_FILTER_MASK_ALL (C macro), 871
WIFI_PROMIS_FILTER_MASK_CTRL (C macro), 871
WIFI_PROMIS_FILTER_MASK_DATA (C macro), 871
WIFI_PROMIS_FILTER_MASK_DATA_AMPDU (C macro), 871
WIFI_PROMIS_FILTER_MASK_DATA_MPDU (C macro), 871
WIFI_PROMIS_FILTER_MASK_FCSFAIL (C macro), 871
WIFI_PROMIS_FILTER_MASK_MGMT (C macro), 871
WIFI_PROMIS_FILTER_MASK_MISC (C macro), 871
wifi_promiscuous_cb_t (C++ type), 853
wifi_promiscuous_filter_t (C++ struct), 863
wifi_promiscuous_filter_t::filter_mask (C++ member), 863
wifi_promiscuous_pkt_t (C++ struct), 863
wifi_promiscuous_pkt_t::payload (C++ member), 863
wifi_promiscuous_pkt_t::rx_ctrl (C++ member), 863
wifi_promiscuous_pkt_t_type_t (C++ enum), 880
wifi_promiscuous_pkt_type_t::WIFI_PKT_CTRL (C++ enumerator), 880
wifi_promiscuous_pkt_type_t::WIFI_PKT_DATA (C++ enumerator), 880
wifi_promiscuous_pkt_type_t::WIFI_PKT_MGMT (C++ enumerator), 880
wifi_promiscuous_pkt_type_t::WIFI_PKT_MISC (C++ enumerator), 880
WIFI_PROTOCOL_11B (C macro), 871
WIFI_PROTOCOL_11G (C macro), 871
WIFI_PROTOCOL_11N (C macro), 871
WiFi_prov_cb_event_t (C++ enum), 1634
WiFi_prov_cb_event_t::WIFI_prov_CRED_FAIL (C++ enumerator), 1634
WiFi_prov_cb_event_t::WIFI_prov_CRED_FAILV (C++ enumerator), 1634
WiFi_prov_cb_event_t::WIFI_prov_CRED_RECV (C++ enumerator), 1634
WiFi_prov_cb_event_t::WIFI_prov_CRED_SUCCESS (C++ enumerator), 1635
WiFi_prov_cb_event_t::WIFI_prov_DEINIT (C++ enumerator), 1635
WiFi_prov_cb_event_t::WIFI_prov_END (C++ enumerator), 1635
WiFi_prov_cb_event_t::WIFI_prov_INIT (C++ enumerator), 1634
WiFi_prov_cb_event_t::WIFI_prov_START (C++ enumerator), 1634
wifi_prov_cb_func_t (C++ type), 1634
wifi_prov_config_data_handler (C++ function), 1637
wifi_prov_config_get_data_t (C++ struct), 1637
wifi_prov_config_get_data_t::conn_info (C++ member), 1637
wifi_prov_config_get_data_t::fail_reason (C++ member), 1637
wifi_prov_config_get_data_t::wifi_state (C++ member), 1637
wifi_prov_config_handlers (C++ struct), 1638
wifi_prov_config_handlers::apply_config_wifi_prov_mgr_stop_provisioning (C++ function), 1628
wifi_prov_config_handlers::ctx (C++ member), 1638
wifi_prov_config_handlers::get_status_wifi_prov_mgr_start_provisioning (C++ function), 1629
wifi_prov_config_handlers::set_config_wifi_prov_mgr_stop_provisioning (C++ function), 1628
wifi_prov_config_handlers_t (C++ type), 1638
wifi_prov_config_set_data_t (C++ struct), 1638
wifi_prov_config_set_data_t::bssid (C++ member), 1638
wifi_prov_config_set_data_t::channel (C++ member), 1638
wifi_prov_config_set_data_t::password (C++ member), 1638
wifi_prov_config_set_data_t::ssid (C++ member), 1638
wifi_prov_ctxt_t (C++ type), 1638
WIFI_PROV_EVENT_HANDLER_NONE (C macro), 1634
wifi_prov_event_handler_t (C++ struct), 1632
wifi_prov_event_handler_t::event_cb (C++ member), 1632
wifi_prov_event_handler_t::user_data (C++ member), 1632
wifi_prov_mgr_config_t (C++ struct), 1633
wifi_prov_mgr_config_t::app_event_handler (C++ member), 1633
wifi_prov_mgr_config_t::scheme (C++ member), 1633
wifi_prov_mgr_config_t::scheme_event_handler (C++ member), 1633
wifi_prov_mgr_configure_sta (C++ function), 1631
wifi_prov_mgr_deinit (C++ function), 1627
wifi_prov_mgr_disable_auto_stop (C++ function), 1629
wifi_prov_mgr_endpoint_create (C++ function), 1630
wifi_prov_mgr_endpoint_register (C++ function), 1630
wifi_prov_mgr_endpoint_unregister (C++ function), 1631
wifi_prov_mgr_get_wifi_disconnect_reason (C++ function), 1631
wifi_prov_mgr_get_wifi_state (C++ function), 1631
wifi_prov_mgr_init (C++ function), 1627
wifi_prov_mgr_is_provisioned (C++ function), 1627
wifi_prov_mgr_reset_provisioning (C++ function), 1631
wifi_prov_mgr_reset_sm_state_for_reprovision (C++ function), 1632
wifi_prov_mgr_reset_sm_state_on_failure (C++ function), 1632
wifi_prov_mgr_set_app_info (C++ function), 1629
wifi_prov_mgr_start_provisioning (C++ function), 1628
wifi_prov_mgr_stop_provisioning (C++ function), 1628
wifi_prov_mgr_wait (C++ function), 1629
wifi_prov_scheme (C++ struct), 1632
wifi_prov_scheme::new_config (C++ member), 1633
wifi_prov_scheme::prov_start (C++ member), 1632
wifi_prov_scheme::prov_stop (C++ member), 1633
wifi_prov_scheme::set_config_endpoint (C++ member), 1633
wifi_prov_scheme::set_config_service (C++ member), 1633
wifi_prov_scheme::wifi_mode (C++ member), 1633
wifi_prov_scheme_ble_event_cb_free_ble (C++ function), 1635
wifi_prov_scheme_ble_event_cb_free_bt (C++ function), 1635
wifi_prov_scheme_ble_event_cb_free_btdm (C++ function), 1635
wifi_prov_scheme_ble_event_handler_free_BLE (C macro), 1636
wifi_prov_scheme_ble_event_handler_free_BT (C macro), 1636
wifi_prov_scheme_ble_event_handler_free_BTDM (C macro), 1636
wifi_prov_scheme_ble_set_mfg_data (C++ function), 1636
wifi_prov_scheme_ble_set_service_uuid (C++ function), 1635
wifi_prov_scheme_software_set_httpd_handle (C++ function), 1636
wifi_prov_scheme_t (C++ type), 1634
wifi_prov_security (C++ enum), 1635
wifi_prov_security2_params_t (C++ type),
wifi_sta_config_t::password (C++ member), 858
wifi_sta_config_t::pmf_cfg (C++ member), 859
wifi_sta_config_t::reserved (C++ member), 859
wifi_sta_config_t::rm_enabled (C++ member), 859
wifi_sta_config_t::sae_pwe_h2e (C++ member), 859
wifi_sta_config_t::scan_method (C++ member), 859
wifi_sta_config_t::sort_method (C++ member), 859
wifi_sta_config_t::ssid (C++ member), 858
wifi_sta_config_t::threshold (C++ member), 859
WIFI_STA_DISCONNECTED_PM_ENABLED (C macro), 853
wifi_sta_info_t (C++ struct), 860
wifi_sta_info_t::is_mesh_child (C++ member), 860
wifi_sta_info_t::mac (C++ member), 860
wifi_sta_info_t::phy_11b (C++ member), 860
wifi_sta_info_t::phy_11g (C++ member), 860
wifi_sta_info_t::phy_11n (C++ member), 860
wifi_sta_info_t::phy_lr (C++ member), 860
wifi_sta_info_t::reserved (C++ member), 860
wifi_sta_list_t::rsid (C++ member), 860
WIFI_STATS_ALL (C macro), 852
WIFI_STATS_ALL_TX (C macro), 873
WIFI_STATS_ALL_RX (C macro), 873
WIFI_STATIS_HW (C macro), 873
WIFI_STATIS_PS (C macro), 873
WIFI_STATIS_RXTX (C macro), 873
wifi_storage_t (C++ enum), 879
wifi_storage_t::WIFI_STORAGE_FLASH (C++ enumerator), 879
wifi_storage_t::WIFI_STORAGE_RAM (C++ enumerator), 880
WIFI_TASK_CORE_ID (C macro), 853
WIFI_VENDOR_IE_ELEMENT_ID (C macro), 871
wifi_vendor_ie_id_t (C++ enum), 880
wifi_vendor_ie_id_t::WIFI_VND_IE_ID_0 (C++ enumerator), 880
wifi_vendor_ie_id_t::WIFI_VND_IE_ID_1 (C++ enumerator), 880
wifi_storage_t::WIFI_VND_IE_TYPE_ASSOC_REQ (C++ enumerator), 880
wifi_vendor_ie_type_t::WIFI_VND_IE_TYPE_ASSOC_REQ (C++ enumerator), 880
wifi_vendor_ie_type_t::WIFI_VND_IE_TYPE_PROBE_REQ (C++ enumerator), 880
wifi_vendor_ie_type_t::WIFI_VND_IE_TYPE_PROBE_RESP (C++ enumerator), 880
wl_erase_range (C++ function), 1748
wl_handle_t (C++ type), 1749
wl_INTEGRAL_HANDLE (C macro), 1749
wl.mount (C++ function), 1748
wl_read (C++ function), 1749
wl_sector_size (C++ function), 1749
wl_size (C++ function), 1749
wl_unmount (C++ function), 1748
wl_write (C++ function), 1748

X
xEventGroupClearBits (C++ function), 1905
xEventGroupClearBitsFromISR (C macro), 1908
xEventGroupCreate (C++ function), 1902
xEventGroupCreateStatic (C++ function), 1903
xEventGroupGetBits (C macro), 1910
xEventGroupGetBitsFromISR (C++ function), 1908
xEventGroupSetBits (C++ function), 1905
xEventGroupSetBitsFromISR (C macro), 1909
xEventGroupSync (C++ function), 1906
xEventGroupWaitBits (C++ function), 1903
xMessageBufferCreate (C macro), 1919
xMessageBufferCreateStatic (C macro), 1919
xMessageBufferIsEmpty (C macro), 1925
xMessageBufferIsFull (C macro), 1925
xMessageBufferNextLengthBytes (C macro), 1926
xMessageBufferReceive (C macro), 1923
xMessageBufferReceiveCompletedFromISR (C macro), 1926
xMessageBufferReceiveFromISR (C macro), 1924
xMessageBufferReset (C macro), 1925
xMessageBufferSend (C macro), 1920
xMessageBufferSendCompletedFromISR (C macro), 1926
xMessageBufferSpacesAvailable (C macro), 1926
xMessageBufferSpacesAvailable (C macro), 1926
xQueueAddToSet (C++ function), 1858
xQueueCreate (C macro), 1859
xQueueCreateSet (C++ function), 1858
xQueueCreateStatic (C macro), 1860
xQueueGenericCreate (C++ function), 1857
xQueueGenericCreateStatic (C function), 1857
xQueueGenericSend (C function), 1851
xQueueGenericSendFromISR (C function), 1855
xQueueGiveFromISR (C function), 1856
xQueueIsQueueEmptyFromISR (C function), 1857
xQueueIsQueueFullFromISR (C function), 1857
xQueueOverwrite (C macro), 1864
xQueueOverwriteFromISR (C macro), 1867
xQueuePeek (C function), 1852
xQueuePeekFromISR (C function), 1853
xQueueReceive (C function), 1853
xQueueReceiveFromISR (C function), 1856
xQueueRemoveFromSet (C function), 1858
xQueueReset (C macro), 1869
xQueueSelectFromSet (C function), 1859
xQueueSelectFromSetFromISR (C function), 1859
xQueueSend (C macro), 1863
xQueueSendFromISR (C macro), 1868
xQueueSendToBack (C macro), 1862
xQueueSendToBackFromISR (C macro), 1866
xQueueSendToFront (C macro), 1861
xQueueSendToFrontFromISR (C macro), 1865
xRingbufferAddToQueueSetRead (C function), 1943
xRingbufferCanRead (C function), 1943
xRingbufferCreate (C function), 1937
xRingbufferCreateNoSplit (C function), 1937
xRingbufferCreateStatic (C function), 1937
xRingbufferGetCurFreeSize (C function), 1942
xRingbufferGetMaxItemSize (C function), 1942
xRingbufferPrintInfo (C function), 1944
xRingbufferReceive (C function), 1939
xRingbufferReceiveFromISR (C function), 1939
xRingbufferReceiveSplit (C function), 1940
xRingbufferReceiveSplitFromISR (C function), 1940
xRingbufferReceiveUpTo (C function), 1941
xRingbufferReceiveUpToFromISR (C function), 1941
xRingbufferRemoveFromQueueSetRead (C function), 1943
xRingbufferSend (C function), 1938
xRingbufferSendAcquire (C function), 1938
xRingbufferSendComplete (C function), 1939
xRingbufferSendFromISR (C function), 1938
xSemaphoreCreateBinary (C macro), 1870
xSemaphoreCreateBinaryStatic (C macro), 1871
xSemaphoreCreateCounting (C macro), 1879
xSemaphoreCreateCountingStatic (C macro), 1882
xSemaphoreCreateMutex (C macro), 1878
xSemaphoreCreateMutexStatic (C macro), 1879
xSemaphoreGetMutexHolder (C macro), 1883
xSemaphoreGetMutexHolderFromISR (C macro), 1883
xSemaphoreGive (C macro), 1874
xSemaphoreGiveFromISR (C macro), 1876
xSemaphoreGiveRecursive (C macro), 1875
xSemaphoreTake (C macro), 1872
xSemaphoreTakeFromISR (C macro), 1878
xSemaphoreTakeRecursive (C macro), 1873
xSTATIC_RINGBUFFER (C++ struct), 1944
xStreamBufferBytesAvailable (C function), 1915
xStreamBufferCreate (C macro), 1917
xStreamBufferCreateStatic (C macro), 1917
xStreamBufferIsEmpty (C function), 1915
xStreamBufferIsFull (C function), 1915
xStreamBufferReceive (C function), 1913
xStreamBufferReceiveCompletedFromISR (C function), 1916
xStreamBufferReceiveFromISR (C function), 1914
xStreamBufferReset (C function), 1915
xStreamBufferSend (C function), 1910
xStreamBufferSendCompletedFromISR (C function), 1916
xStreamBufferSendFromISR (C function), 1911
xStreamBufferSetTriggerLevel (C function), 1915
xStreamBufferSpacesAvailable (C function), 1915
xTaskAbortDelay (C function), 1826
xTaskCallApplicationTaskHook (C function), 1835
xTaskCatchUpTicks (C function), 1846
xTaskCheckForTimeOut (C function), 1845
xTaskCreate (C function), 1818
xTaskCreatePinnedToCore (C function), 1815
xTaskCreateRestricted (C function), 1821
xTaskCreateStatic (C function), 1819
xTaskCreateStaticPinnedToCore (C function), 1817
xTaskDelayUntil (C function), 1825
xTaskGenericNotify (C function), 1838
xTaskGenericNotifyFromISR (C function), 1839
xTaskGenericNotifyStateClear (C function), 1844
xTaskGenericNotifyWait (C function), 1841
xTaskGetApplicationTaskTag (C++ function), 1834
xTaskGetApplicationTaskTagFromISR (C++ function), 1834
xTaskGetHandle (C++ function), 1833
xTaskGetIdleTaskHandle (C++ function), 1835
xTaskGetTickCount (C++ function), 1832
xTaskGetTickCountFromISR (C macro), 1832
xTaskGetIdleTaskHandle (C++ function), 1835
xTaskGetHandle (C++ function), 1833
xTaskGetTickCountFromISR (C++ function), 1832
xTaskNotify (C macro), 1848
xTaskNotifyAndQuery (C macro), 1848
xTaskNotifyAndQueryFromISR (C macro), 1848
xTaskNotifyAndQueryIndexed (C macro), 1848
xTaskNotifyAndQueryIndexedFromISR (C macro), 1848
xTaskNotifyFromISR (C macro), 1848
xTaskNotifyGive (C macro), 1849
xTaskNotifyGiveIndexed (C macro), 1848
xTaskNotifyIndexed (C macro), 1848
xTaskNotifyIndexedFromISR (C macro), 1848
xTaskNotifyStateClear (C macro), 1849
xTaskNotifyGiveIndexed (C macro), 1849
xTaskNotifyIndexed (C macro), 1848
xTimerChangePeriodFromISR (C macro), 1899
xTimerCreate (C++ function), 1884
xTimerCreateStatic (C++ function), 1886
xTimerDelete (C macro), 1895
xTimerGetExpireTime (C++ function), 1891
xTimerGetPeriod (C++ function), 1891
xTimerGetTimerDaemonTaskHandle (C++ function), 1889
xTimerIsTimerActive (C++ function), 1888
xTimerPendFunctionCall (C++ function), 1890
xTimerPendFunctionCallFromISR (C++ function), 1889
xTimerReset (C macro), 1895
xTimerResetFromISR (C macro), 1900
xTimerStart (C macro), 1892
xTimerStartFromISR (C macro), 1897
xTimerStop (C macro), 1893
xTimerStopFromISR (C macro), 1898

Espressif Systems

2736  Release v5.1-dev-2066-g7869f4e151

Submit Document Feedback